

LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84

I l 6r

no. 782-787

cop. 2



The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

JUN 6 1973

MAY 9 RECD



Digitized by the Internet Archive
in 2013

<http://archive.org/details/designofirredund784yama>

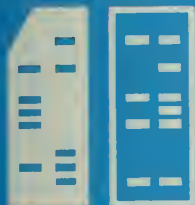
070.84
I 262
Math
Report No. UIUCDCS-R-76-784

DESIGN OF IRREDUNDANT MOS NETWORKS:
A PROGRAM MANUAL FOR THE DESIGN
ALGORITHM DIMN

by

Kazuhiko Yamamoto

Ms. 784
February 1976



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

Report No. UIUCDCS-R-76-784

DESIGN OF IRREDUNDANT MOS NETWORKS:
A PROGRAM MANUAL FOR THE DESIGN
ALGORITHM DIMN

by

Kazuhiko Yamamoto

February 1976

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

This work was supported in part by the National Science Foundation under Grant No. GJ-40221 and was submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, 1976.

ACKNOWLEDGEMENT

The author is greatly appreciative of Professor S. Muroga for his discussions and guidance relating to the preparation of this paper, and also, for his careful reading and variable suggestions for the improvement of the original manuscript. The author is also obliged to J. N. Culliney and K. C. Hu for their indispensable guidance and stimulating discussions.

This work was supported in part by the Department of Computer Science, University of Illinois and also the National Science Foundation under Grant No. GJ-40221.

TABLE OF CONTENTS

CHAPTER	Page
1. INTRODUCTION.....	1
2. THEORETICAL BACKGROUND.....	3
3. PROCEDURE DIMN.....	14
3.1 Internal Data Representation.....	15
3.2 General Organization of Program DIMN.....	22
3.3 Description About Subroutines.....	26
4. INPUT DATA SETUP.....	70
4.1 Input Data Card Format.....	70
4.2 Restriction on Problem Size.....	73
4.3 Example of Input Data Setup.....	75
5. OUTPUT OF PROCEDURE DIMN.....	78
5.1 Output Format.....	78
5.2 Networks Obtained by Program DIMN.....	81
6. CONCLUSION.....	85
REFERENCES.....	86
APPENDICES	
A - Networks Obtained by Program DIMN.....	87
B - Program Listing.....	103

1. INTRODUCTION

With recent progress in integrated circuit technology, MOS logic circuit has become one of the most important logic families for digital computers. MOS logic circuit has a lot of advantages over bipolar devices such as high packing density, lower power consumption and simple production process.

Since at least theoretically a MOS cell can realize an arbitrary negative function, several algorithms for designing logic circuit based on MOS cell's capability of expressing any negative gate have been developed. An algorithm which derives a two level network of a minimum number of negative gates was first developed by Ibaraki and Muroga [1] [2]. Then efficient algorithms which can also realize multilevel network with minimum number of negative gates[†] were developed by Liu [3] and Kasami et al [4].

Although the algorithms mentioned above guarantee the minimality of the number of MOS cells, the network synthesized by these algorithms may still have redundant connections and MOS FETs. Recently, H. C. Lai [5] has modified those algorithms and introduced a new algorithm called DIMN (Design of Irredundant MOS Network) which finds irredundant MOS networks with a minimum number of MOS cells for a given set of functions.

In this paper, a FORTRAN program package DIMN which designs MOS logic networks based on Lai's algorithm is described. This program DIMN

[†]In the case of MOS network, we usually use the term "cell" instead of "gates."

is applicable to networks with multiple incompletely specified output functions. Consequently, it covers almost all the algorithms introduced in Lai's paper. In other words, this program has the capability to realize networks with a completely specified single output function, networks with completely specified multiple output functions and networks with an incompletely specified single output function. The only exception is the network with all the output functions in the output level. As the algorithm for realizing this network requires a slightly different procedure from the procedures for realizing the networks mentioned above, it is not implemented in the program DIMN. Another fact that has to be mentioned here is that the current program DIMN obtains only one irredundant MOS network for any given function. It is hoped that this program will be eventually modified to exhaust all the irredundant MOS networks.

The next chapter, Chapter 2, is allocated for the review of Algorithm DIMN. In Chapter 3, the program DIMN is discussed in greater detail. Chapter 4 outlines the preparation of the input for this program. Chapter 5 describes the output of program DIMN and also compares the networks obtained by applying Algorithm DIMN with the networks obtained by Liu's algorithm. Finally the networks obtained by program DIMN for several three and four variable functions and a complete listing of FORTRAN program DIMN are given in Appendices A and B, respectively.

2. THEORETICAL BACKGROUND

This section reviews Lai's Algorithm (Algorithm DIMN). Like those previously developed algorithms, Algorithm DIMN repeats two phases, that is, phase 1, the derivation of a function for each negative gate (MOS cell) and phase 2, the design of an irredundant MOS cell configuration for the function obtained in phase 1. The repetition of these two phases leads to the design of an entire network. The difference between Algorithm DIMN and the previously developed algorithms can be seen in how these two steps are implemented. In the previously developed algorithms, the implementation is accomplished with a single pass application of phase 1 followed by phase 2. On the contrary, in Algorithm DIMN, these two phases are applied interactively to guarantee the irredundancy of the network.

The simplest case of Algorithm DIMN i.e., the one for obtaining a network with a completely specified single output function is shown in the following. This simplest version of Algorithm DIMN can be easily extended to the cases for a network with an incompletely specified single output function, for a network with completely specified multiple output functions and for a network with incompletely specified multiple output functions. In order to facilitate our discussion, some notations and terminologies are explained (see Lai's thesis for detail).

Our design objective is to obtain a loopless network which consists of negative gates only for a given function f . The negative gate is a

gate which realizes a switching function which can be expressed in the form of the complement of a disjunctive form with non-complemented literals only. A generalized form of a loopless network with R_f negative gates is shown in Fig. 2.1, where x_1, \dots, x_N denote N external variables and u_1, \dots, u_{R_f} denote the functions realized by the negative gates g_1, \dots, g_{R_f} in the network.

$$\text{NFS } (R_f, f) = (u_1, \dots, u_{R_f-1}, f):$$

In this generalized form of a loopless network, the sequence of functions u_1, \dots, u_{R_f} is called a negative function sequence of length R_f for a function f and is denoted by $\text{NFS } (R_f, f) = (u_1, \dots, u_{R_f-1}, f)$.

$$\text{NFS}^i (R_f, f) = (u_1, \dots, u_i, u_{i+1}^*, \dots, u_{R_f-1}^*, f):$$

This represents a partially specified negative function sequence of length R_f and degree i for a function f . Unlike the previously mentioned $\text{NFS } (R_f, f)$, the functions in this $\text{NFS}^i (R_f, f)$ are not completely specified. (In $\text{NFS}^i (R_f, f)$, first i functions with no $*$ are completely specified, but the remaining $R_f - i - 1$ functions with $*$ are unspecified.) A completion of $\text{NFS}^i (R_f, f)$ is a function sequence obtained by completely specifying the unspecified functions in $\text{NFS}^i (R_f, f)$.

N-cube: This is a lattice which represents switching functions with N external variables. In the N -cube, each vertex corresponds to an input vector to the functions and the vertices of the same weight

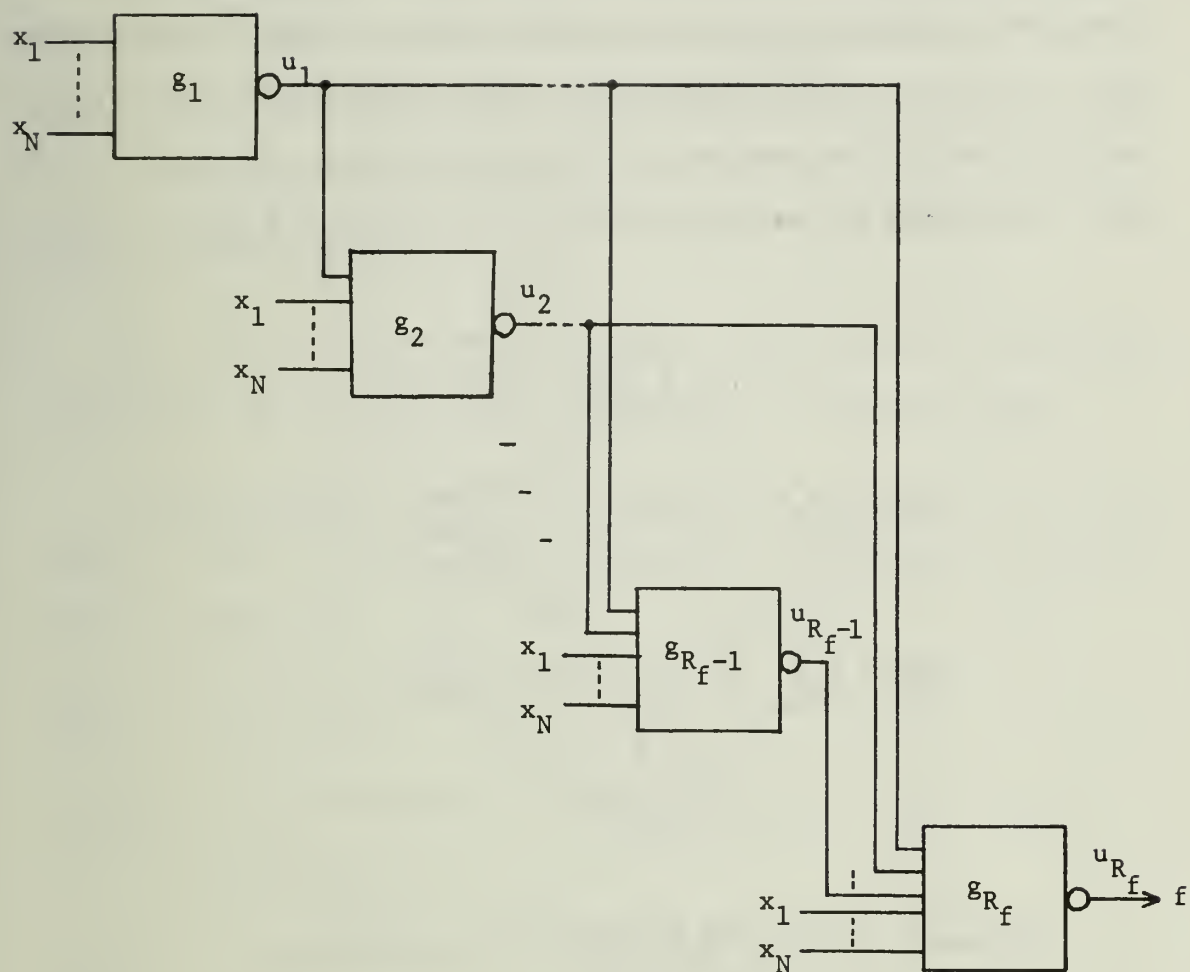


Fig. 2.1 Generalized form of a loopless network consisting of R_f negative gates.

(number of ones in an input vector assigned to the vertex) are in the same level, placing vertices with more weight in a higher level. Every pair of vertices which corresponds to input vectors differing in only one bit position is connected by an edge. An example is shown in Fig. 2.2 for $N=3$ and two functions ($f_1 = \bar{x}_1 \vee \bar{x}_3$, $f_2 = \bar{x}_1 x_2 \bar{x}_3$).

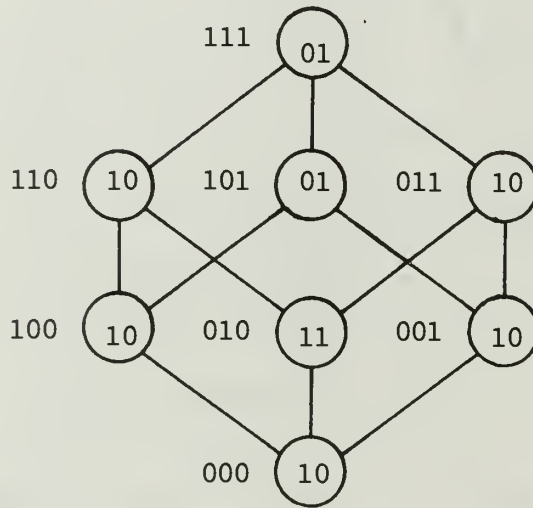


Fig. 2.2 3-cube for 2 functions; $f_1 = \bar{x}_1 \vee \bar{x}_3$,
 $f_2 = x_1 x_3 \vee \bar{x}_1 x_2 \bar{x}_3$.

Algorithm CMNL: Algorithm CMNL (Conditional Minimum Labeling) obtains $\text{NFS}^{i-1}(R_f, f) = (u_1, \dots, u_{i-1}, \underline{u}_i, \dots, \underline{u}_{R_f-1}, f)$ which is the completion of $\text{NFS}^{i-1}(R_f, f)$ such that the label assigned to each vertex in the N -cube takes the minimum possible value of all feasible completions of $\text{NFS}^{i-1}(R_f, f)$. The feasible completion is a completion such that the resulting N -cube has no inverse edge.

Algorithm CMXL: Algorithm CMXL (Conditional Maximum Labeling)

obtains $\hat{NFS}^{i-1}(R_f, f) = (u_1, \dots, u_{i-1}, \hat{u}_i, \dots, \hat{u}_{R_f-1}, f)$ which is the completion of $NFS^{i-1}(R_f, f)$ such that the label assigned to each vertex in the N-cube takes the maximum possible value of all feasible completions of $NFS^{i-1}(R_f, f)$.

\hat{u}_i : This is the maximum permissible function which is obtained in the process which will be described in Step 4 of Algorithm DIMN.

Algorithm DIMN: Design of irredundant MOS networks with a minimum number of MOS cells for a given function f . R_f represents the minimum number of MOS cells.

Step 1 Let $NFS^0(R_f, f) = (u_1^*, \dots, u_{R_f-1}^*, f)$ and set $i=1$

Step 2 Use algorithm CMNL to obtain $\underline{NFS}^{i-1}(R_f, f) = (u_1, \dots, u_{i-1}, \underline{u}_i, \dots, \underline{u}_{R_f-1}, f)$

Step 3 Use algorithm CMXL to obtain $\hat{NFS}^{i-1}(R_f, f) = (u_1, \dots, u_{i-1}, \hat{u}_i, \dots, \hat{u}_{R_f-1}, f)$

Step 4 Obtain function \hat{u}_i by setting

$$\hat{u}_i(A) = 0, \text{ if } \underline{u}_i(A) = \hat{u}_i(A) = 0$$

$$\hat{u}_i(A) = 1, \text{ if } \underline{u}_i(A) = \hat{u}_i(A) = 1$$

$$\hat{u}_i(A) = *, \text{ if } \underline{u}_i(A) = 0 \text{ and } \hat{u}_i(A) = 1$$

Step 5 Obtain an irredundant MOS cell configuration for \hat{u}_i with respect to $x_1, \dots, x_n, u_1, \dots, u_{i-1}$. Let u_i denote the function realized by this MOS cell (u_i is now a completion of \hat{u}_i).

Step 6 If $i = R_f - 1$, design an irredundant MOS cell configuration for f with respect to $x_1, \dots, x_n, u_1, \dots, u_{R_f-1}$ and terminate this algorithm. Otherwise set $i = i+1$ and go to Step 2.

It should be noted that in the above Algorithm DIMN, Steps 2, 3 and 4 are deterministic, in other words, given a $NFS^{i-1}(R_f, f)$, subsequent $\underline{NFS}^{i-1}(R_f, f)$, $\hat{NFS}^{i-1}(R_f, f)$ and $\check{NFS}^{i-1}(R_f, f)$ are uniquely determined. On the other hand, Step 5 in general is non-deterministic, because more than one irredundant MOS cell configuration may exist for a given \check{u}_i .

Another fact which should be noted is that although Algorithm DIMN requires at most $R_f - 1$ iterations of the loop which consists of Step 2, ..., Step 6 for $i = 1, \dots, R_f - 1$, if $\check{NFS}^{i-1}(R_f, f)$ becomes a completely specified function with respect to x_1, x_2, \dots, x_n for some $i < R_f - 1$, the algorithm actually requires only i iterations. Although, even in this case, Step 5 of the algorithm still has to be executed $R_f - i - 1$ additional times in order to obtain irredundant MOS cell configuration for $u_{i+1}, \dots, u_{R_f-1}$, this fact will help reducing the computation time when the algorithm is implemented in computer program.

In Fig. 2.3, a simple example is shown for better understanding of Algorithm DIMN. In this example, the 3-cube with respect to a function f to be realized is shown in Fig. 2.3(a).

Step 2 and Step 3 of Algorithm DIMN obtains $\underline{NFS}^0(3, f)$ and $\hat{NFS}^0(3, f)$ as shown in (b) and (c), respectively. Then Step 4 compares \underline{u}_1 and \hat{u}_1

and obtains \tilde{u}_1 and $\hat{NFS}^0(3,f)$ as shown in (d). The $NFS^1(3,f)_1 = (\bar{x}_1, u_2^*, f)$ in (e) is obtained by Step 5. Step 5 also obtains other two irredundant MOS cell configurations for \tilde{u}_1 which are shown in (j) and (o). After Step 6, the algorithm returns to Step 2 and obtains $\underline{NFS}^1(3,f)_1$, $\hat{NFS}^1(3,f)_1$, $\check{NFS}^1(3,f)_1$ and $NFS^2(3,f)_1$ as shown in (f), (g), (h) and (i), respectively. Since $R_f = 3$, $NFS^2(3,f)_1$ is a completely specified negative function sequence with respect to x_1 , x_2 and x_3 . To finish the design, Step 6 of the algorithm obtains an irredundant MOS cell configuration for f . As previously mentioned, three irredundant MOS cell configurations $NFS^1(3,f)_1$, $NFS^1(3,f)_2$ and $NFS^1(3,f)_3$ are obtained in (e), (j) and (o), respectively for given \tilde{u}_1 in (d). For two other irredundant MOS cell configurations, $NFS^1(3,f)_2$, $NFS^1(3,f)_3$, we can continue applying Algorithm DIMN in the same way starting from (j) and (o), respectively. The resulting irredundant MOS networks are shown in Fig. 2.4 and Fig. 2.5.

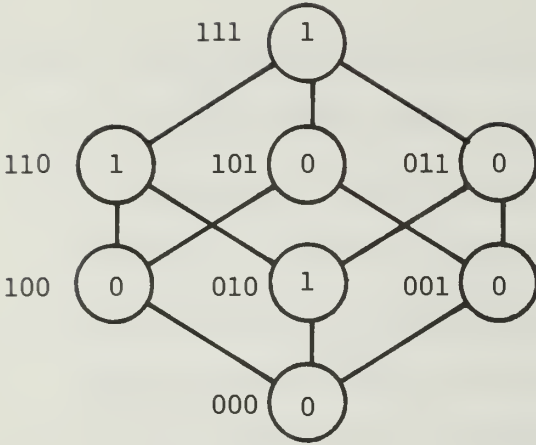
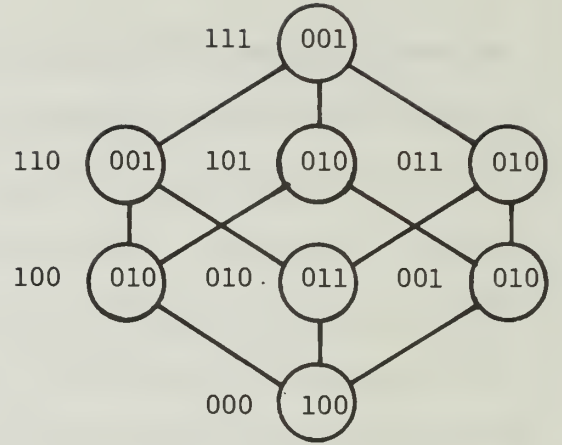
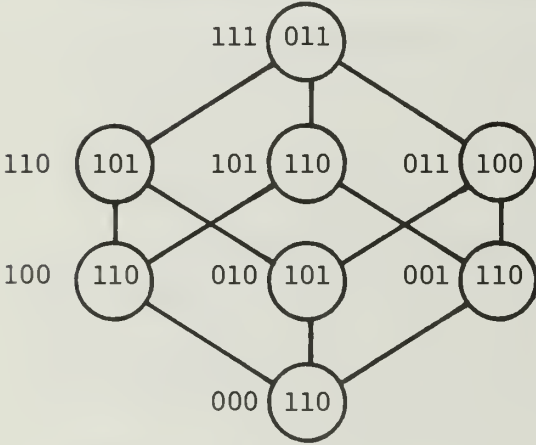
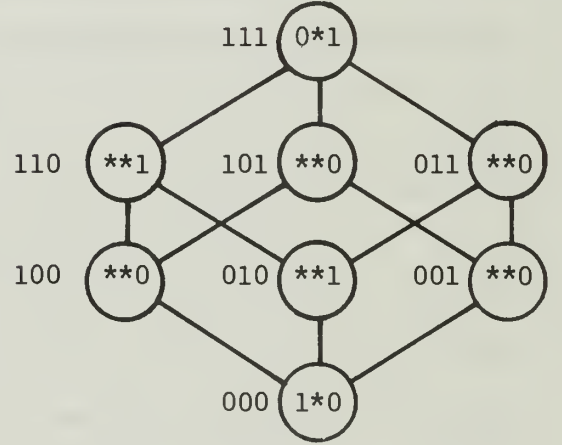
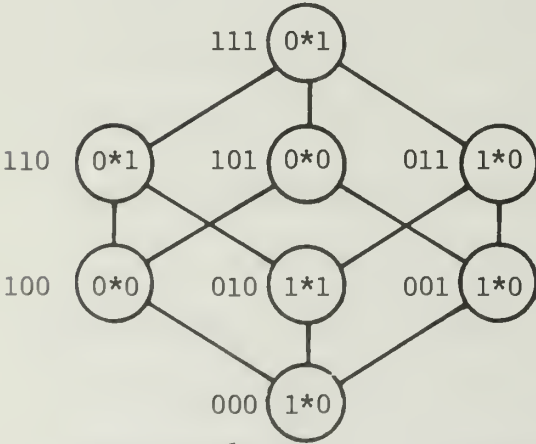
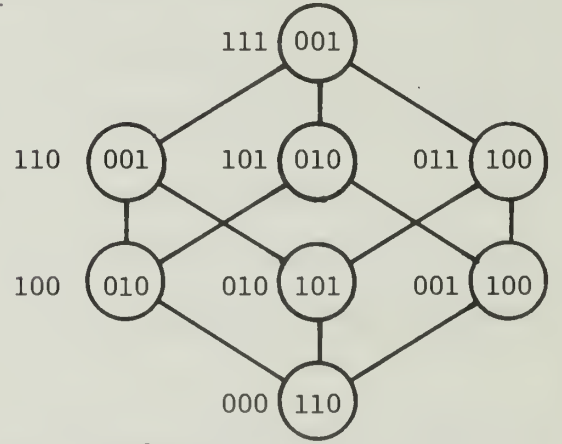
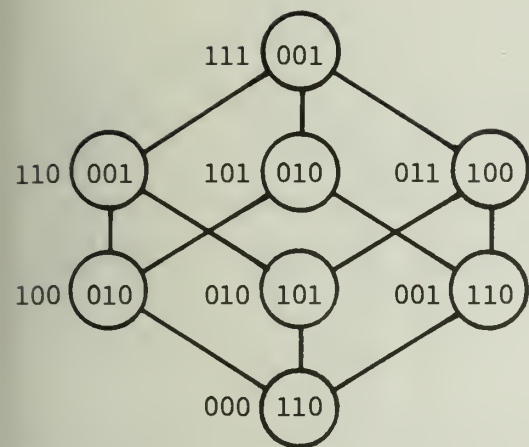
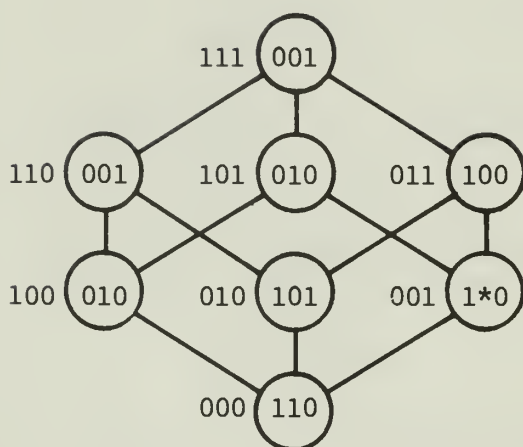
(a) $C_3(f)$ for $f = x_1x_2 \vee x_2\bar{x}_3$ (b) $\underline{NFS}^0(3, f) = (\underline{u}_1, \underline{u}_2, f)$ (c) $\hat{NFS}^0(3, f) = (\hat{u}_1, \hat{u}_2, f)$ (d) $\tilde{NFS}^0(3, f) = (\tilde{u}_1, u_2^*, f)$ (e) $\text{First } NFS^1(3, f)_1 = (\bar{x}_1, u_2^*, f)$ (f) $\underline{NFS}^1(3, f)_1 = (\bar{x}_1, \underline{u}_2, f)$

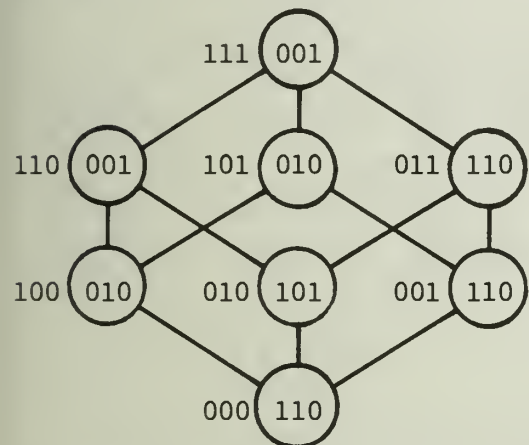
Fig. 2.3 Example for Algorithm DIMN.



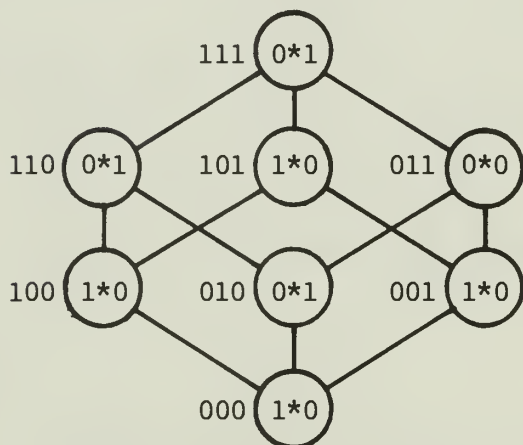
(g) $\hat{NFS}^1(3, f)_1 = (\bar{x}_1, \hat{u}_2, f)$



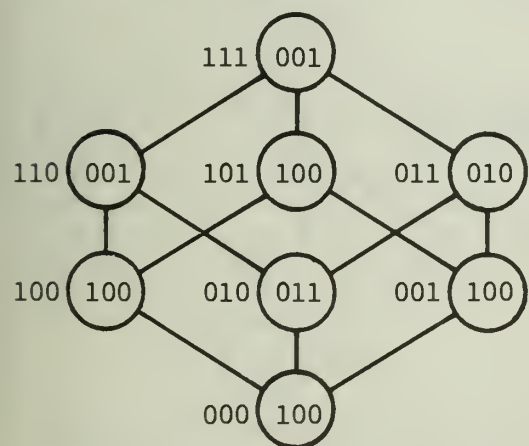
(h) $\hat{NFS}^1(3, f)_1 = (\bar{x}_1, \hat{u}_2, f)$



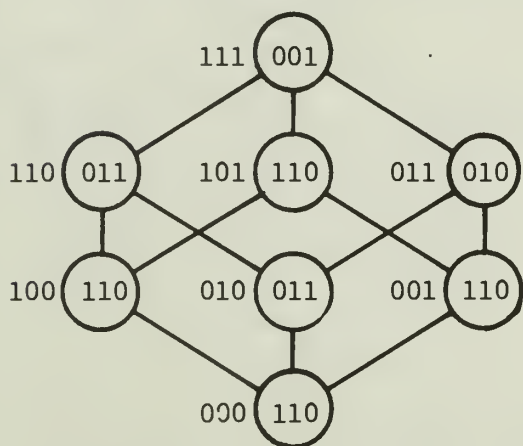
(i) $NFS^2(3, f)_1 = NFS(3, f)_1 = (\bar{x}_1, \bar{x}_2, f)$



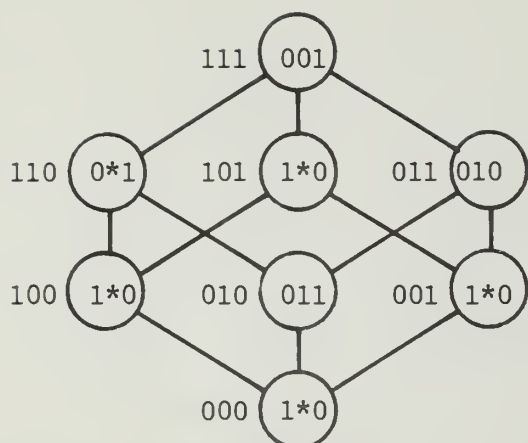
(j) Second $NFS^1(3, f)_2 = (\bar{x}_2, u_2^*, f)$ obtained from (d)



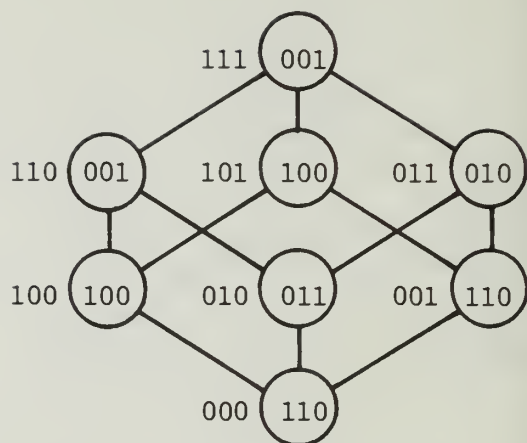
(k) $\underline{NFS}^1(3, f)_2 = (\bar{x}_2, \underline{u}_2, f)$



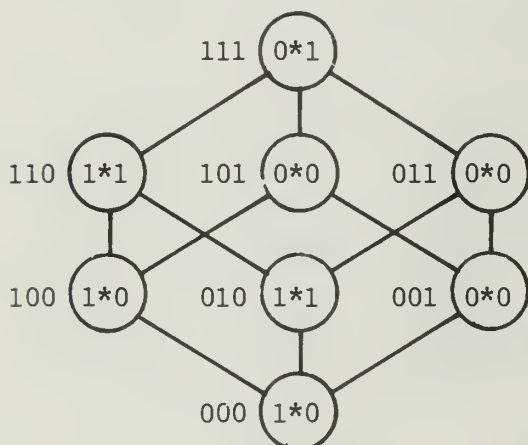
(l) $\hat{NFS}^1(3, f)_2 = (\bar{x}_2, \hat{u}_2, f)$



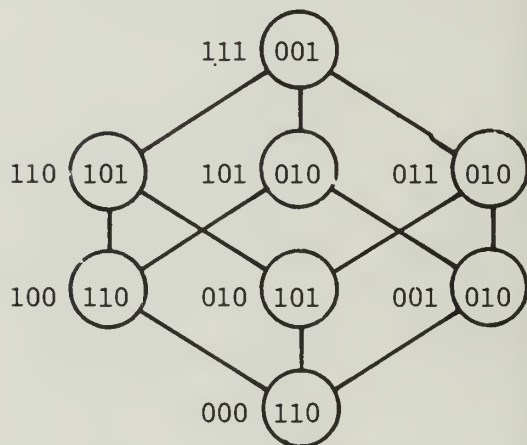
(m) $\hat{\text{NFS}}^1(3, f)_2 = (\bar{x}_2, \hat{u}_2, f)$



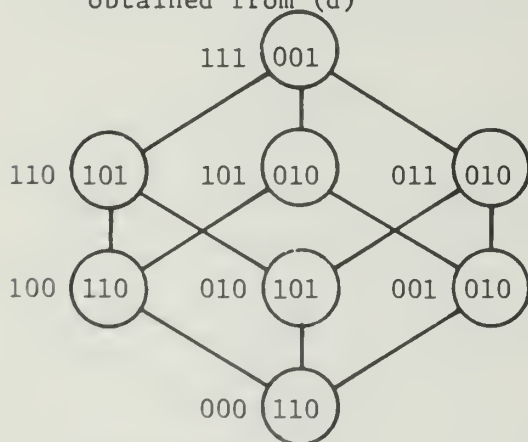
(n) $\text{NFS}^2(3, f)_2 = \text{NFS}(3, f)_2 = (\bar{x}_2, \bar{x}_1, f)$



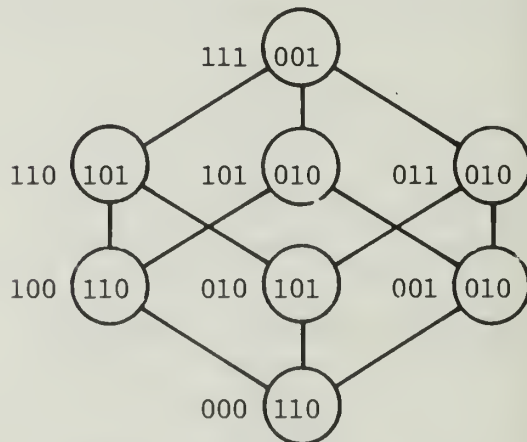
(o) Third $\text{NFS}^1(3, f)_3 = (\bar{x}_3, u_2^*, f)$ obtained from (d)



(p) $\underline{\text{NFS}}^1(3, f)_3 = (\bar{x}_3, u_2, f)$



(q) $\hat{\text{NFS}}^1(3, f)_3 = (\bar{x}_3, \hat{u}_2, f)$



(r) $\hat{\text{NFS}}^1(3, f)_3 = \text{NFS}(3, f) = (\bar{x}_3, \bar{f}, f)$

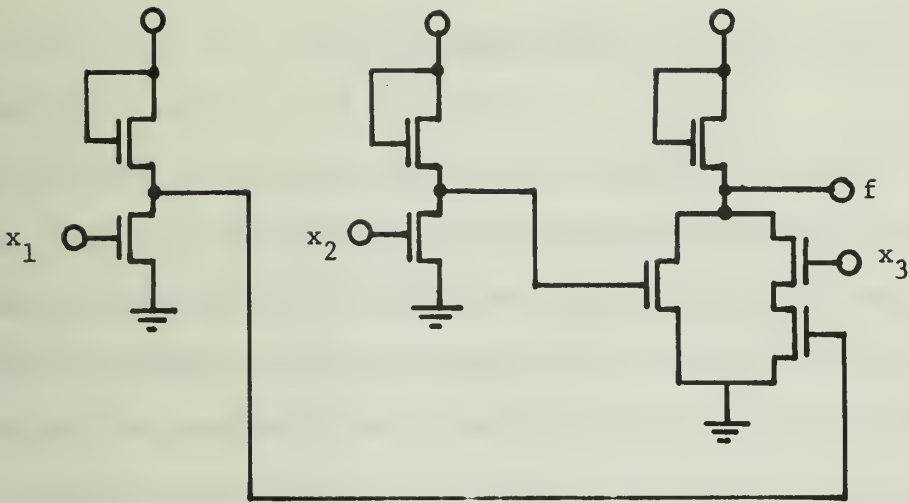


Fig. 2.4 Irredundant MOS network corresponding to NFS(3,f)'s in (i) and (n).

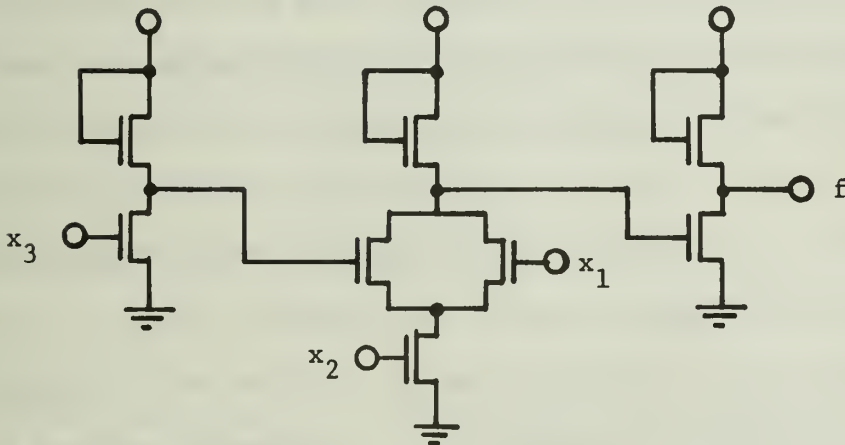


Fig. 2.5 Irredundant MOS network corresponding to NFS(3,f)₃ in (r).

3. PROCEDURE DIMN

This chapter will discuss the design procedure in FORTRAN program DIMN for designing an irredundant MOS network.

An input to this program is a description of output functions in truth table form which the resulting network has to realize. This description (explained in detail in Chapter 4) consists of network parameters and output function description. The output of this program is the description of realized irredundant MOS networks (explained in detail in Chapter 5).

The entire DIMN program requires 168K bites of core storage, about 62K being occupied by the actual program instructions and 106K by the stored data (compiled by FORTRAN G compiler).

For better understanding of the procedure DIMN, Section 3.1 describes internal data representation, that is, the representation of a labeled N-cube (for the definition of this terminology see Chapter 2). Section 3.2 describes the general organization of this program and Section 3.3 explains each subprocedure in more detail. Although the explanation of the variables and arrays appearing in this program can be found in the program listing in the appendix, Section 3.3 also defines some of the variables in order to explain each flowchart.

In the following discussion, N is the number of external variables, M is the number of output functions and RF is the number of MOS cells obtained by this algorithm. II is a pointer which indicates the iteration of the program loop for determining one MOS cell

configuration. For example, if II is three, the third MOS cell configuration is going to be determined.

3.1 Internal Data Representation

In order to implement Algorithm DIMN in a FORTRAN program, a labeled N-cube has to be represented in the computer memory effectively. This section describes how the representation can be accomplished.

To implement the Algorithm DIMN (The flowchart of this algorithm is shown in Fig. 3.1.1. The algorithm shown in Fig. 3.1.1 is slightly modified for networks with incompletely specified multiple output functions.) two kinds of labeled N-cubes are required. One is the labeled N-cube for obtaining MPF (Maximum Permissible Function) and the other is the labeled N-cube for obtaining an irredundant MOS cell configuration. The labeled N-cube for obtaining MPF is accessed in block 2, block 3 and block 4 in Fig. 3.1.1. In the comments of the program listing in the appendix, this labeled N-cube is simply referred to as N-cube and each vertex is assigned a vertex number which represents the input vector in binary form.

As shown in Fig. 3.1.2, each vertex in a N-cube contains the

VERTEX NO.

LABEL	DCARE	MNL	MXL	CHAIN
-------	-------	-----	-----	-------

Fig. 3.1.2 The vertex in the N-cube.

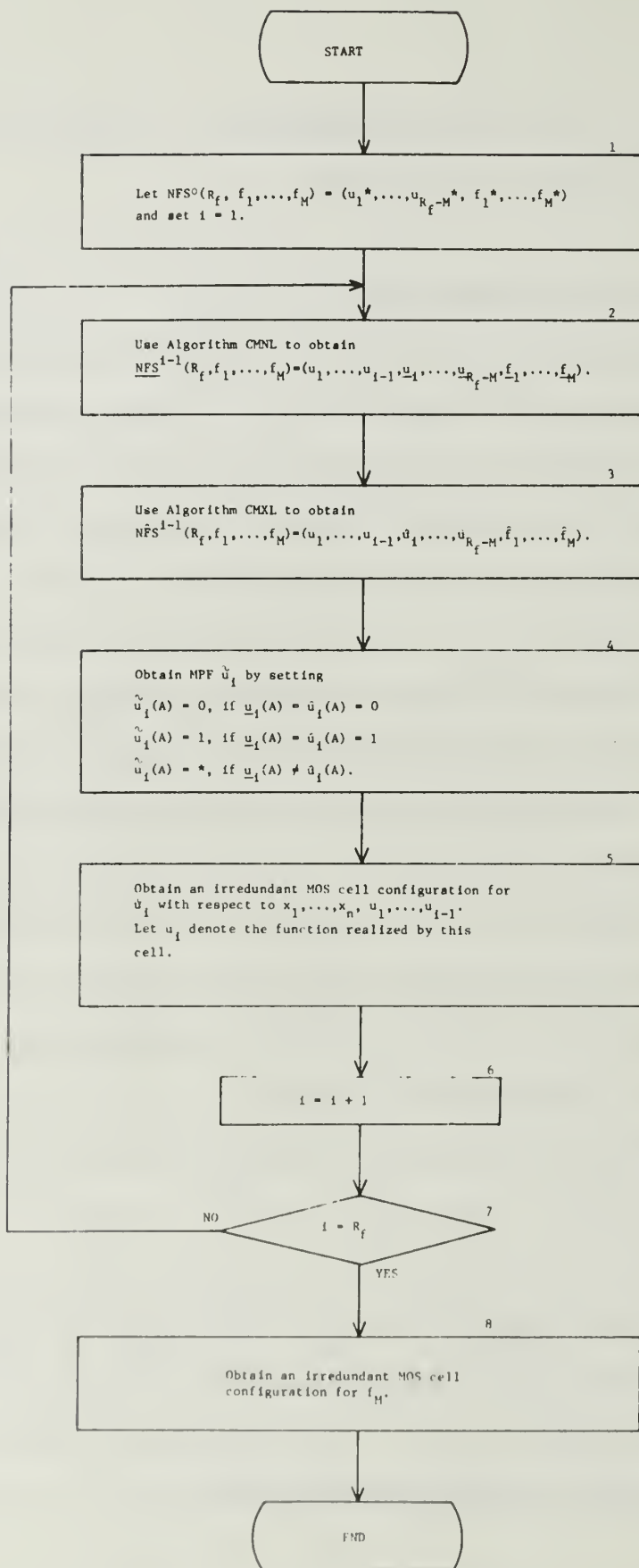


Fig. 3.1.1 Algorithm DIMN.

following five fields; LABEL, DCARE, MNL, MXL and CHAIN. LABEL field, together with DCARE field, stores the partially specified negative function sequence (in Fig. 3.1.1, this is shown with the notation $NFS^{i-1}(R_f, f_1, \dots, f_M)$). MNL and MXL fields store $\underline{NFS}^{i-1}(R_f, f_1, \dots, f_M)$ that is the completion of $NFS^{i-1}(R_f, f_1, \dots, f_M)$ by CMNL, and $\hat{NFS}^{i-1}(R_f, f_1, \dots, f_M)$ that is the completion of $NFS^{i-1}(R_f, f_1, \dots, f_M)$ by CMXL, respectively. CHAIN field stores the link to the next vertex in the list of the vertices of the same weight. Vertex 0, that is, the vertex with vertex number 0 contains the first elements of the arrays LABEL, DCARE, MNL, MXL and CHAIN; vertex 1, that is, the vertex with vertex number 1 contains the second elements of these arrays and so on. In general, the vertex with vertex number (j-1) contains the j-th elements of these arrays as shown in Fig. 3.1.3.

VERTEX (j-1)

LABEL (j)	DCARE (j)	MNL (j)	MXL (j)	CHAIN (j)
-----------	-----------	---------	---------	-----------

Fig. 3.1.3 The vertex with vertex no. (j-1).

Fig. 3.1.4 is an example of N-cube for N=3 and Fig. 3.1.5 is the actual representation of the 3-cube in the computer memory. As can be seen in Fig. 3.1.4, the first and the last vertices in the lists of the vertices with the same weight are pointed by pointers STARTL and ENDL, respectively. For example, vertex 3, the first vertex in the list of

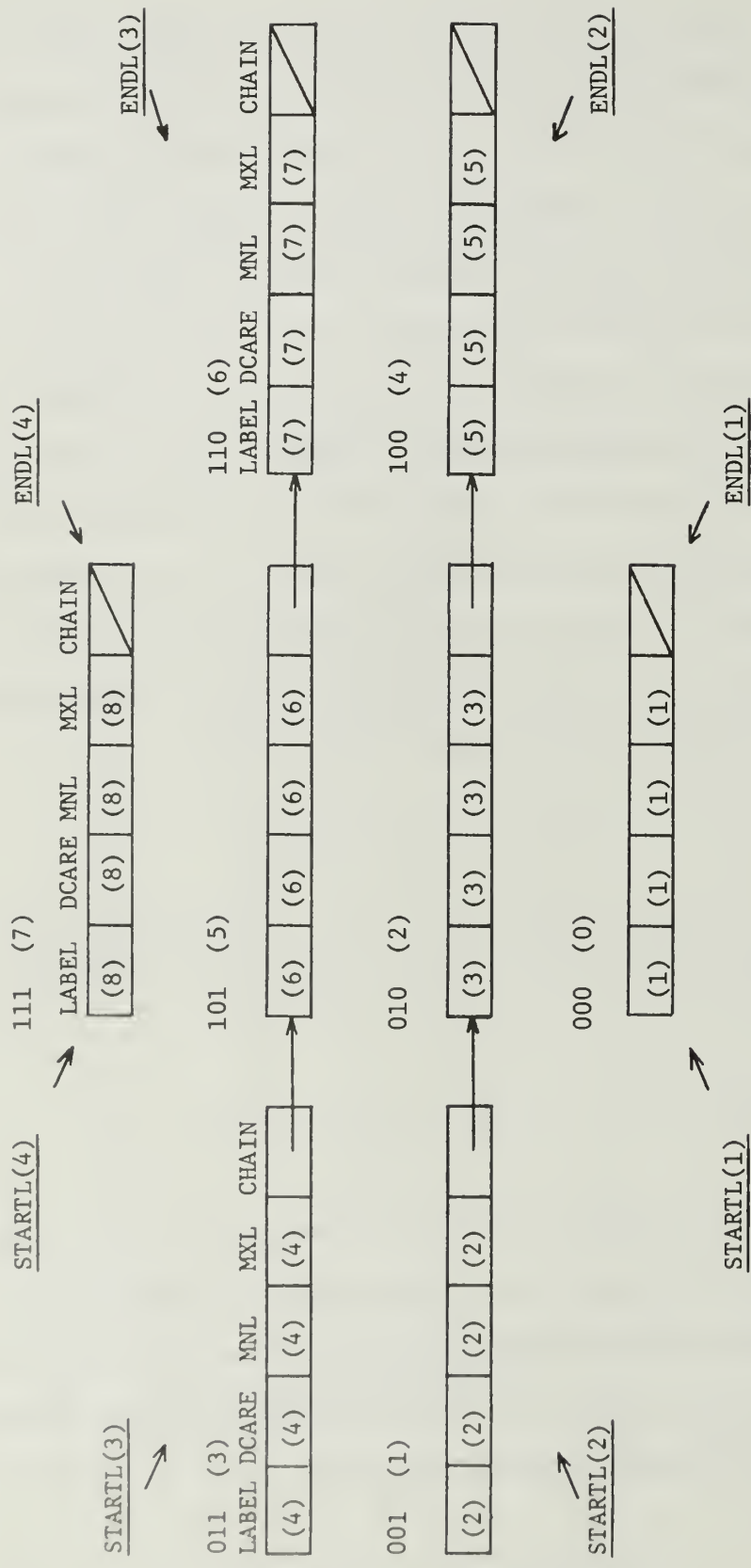


Fig. 3.1.4 3-cube.

	LABEL	DCARE	MNL	MXL	CHAIN
(1)					
(2)					3
(3)					5
(4)					6
(5)					
(6)					7
(7)					
(8)					

	STARTL	ENDL
(1)	1	1
(2)	2	5
(3)	4	7
(4)	8	8

Fig. 3.1.5 Internal representation of the 3-cube in Fig. 3.1.4.

the vertices with weight 2 is pointed by pointer STARTL(3) and vertex 6, the last vertex in the same list is pointed by pointer ENDL(3). In general, the first and the last vertices in the list of the vertices with weight K is pointed by pointers STARTL (k+1) and ENDL (k+1), respectively.

The labeled N-cube for obtaining an irredundant MOS cell configuration is accessed in block 5 in Fig. 3.1.1. In the comments of the program listing in Appendix B, this labeled N-cube is referred to as the Large-cube because the dimension N of this N-cube, starting from the initial value N, is increased by one every time the program loop

in Fig. 3.1.1 is executed. The above discussion implies that this Large-cube has to be newly constructed every time block 5 in Fig. 3.1.1 is executed. On the other hand, the previously mentioned N-cube, once being constructed at the beginning of the program, will never be executed (the contents of the N-cube is changed every time the program loop in Fig. 3.1.1 is executed).

In a similar way to the N-cube, each vertex in the Large-cube which consists of FUNC and LINK fields is assigned a vertex number which is the decimal representation of the corresponding input vector. The LINK field corresponds to the CHAIN field of the N-cube and points to the next vertex with the same weight. MPF (Maximum Permissible Function) is stored in the FUNC field after MPF is obtained at the end of block 4 in Fig. 3.1.1.

At the i -th iteration of the program loop in Fig. 3.1.1, the dimension of this Large-cube will be $N+i-1$ and every time the program loop is executed, the dimension will be increased by one. Therefore, at the last iteration of this program loop (when $i = R_f$), the dimension of this Large-cube will be $N + R_f - 1$. This means that when $N=10$ and $R_f = 5$, the memory space for storing 2^{14} vertices is required in order to store this Large-cube. (The restriction on problem size is due to this memory size. This will be discussed in Section 4.2.)

Fig. 3.1.6 shows an example for 3-dimensional Large-cube (this means $N + i - 1 = 3$) and Fig. 3.1.7 is the internal representation of the 3-dimensional Large-cube shown in Fig. 3.1.6.

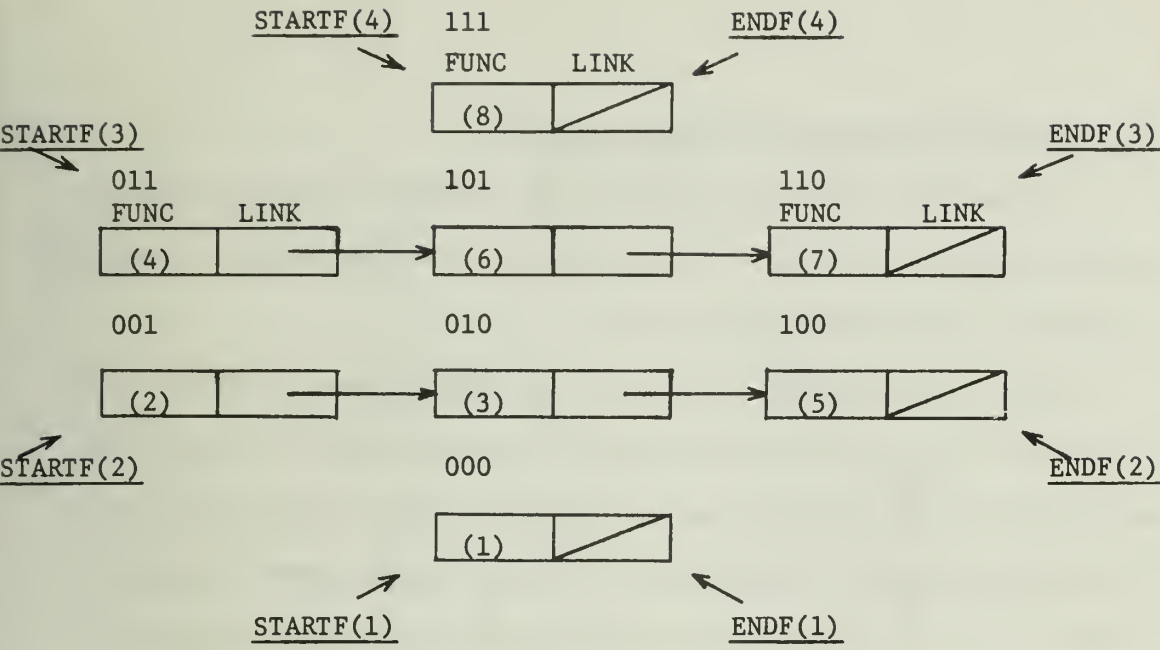


Fig. 3.1.6 3-dimensional Large-cube.

FUNC		LINK	
(1)			
(2)		3	
(3)		5	
(4)		6	
(5)			
(6)		7	
(7)			
(8)			

STARTF		ENDF	
(1)	1	(1)	1
(2)	2	(2)	5
(3)	4	(3)	7
(4)	8	(4)	8

Fig. 3.1.7 Internal representation of 3-dimensional Large-cube in Fig. 3.1.6.

3.2 General Organization of Program DIMN

This section shows the general organization of program DIMN and outlines each subprogram. The function of each subprogram is discussed in detail in the following section.

The general organization of program DIMN is shown in Fig. 3.2.1. As can be seen in this diagram, this program consists of subroutine MAIN and the following subroutines: INPUT, NCUBE, CMNL, CMXL, MPF, IMC, ASFUN1, ASFUN2, INCMNT, DSCAN, ASIGN1, DCRMNT, CMPR and PRINT. In Fig. 3.2.1, an arrow from block i to block j represents the fact that the subroutine represented by block i calls the subroutine represented by block j.

Subroutine INPUT: This subroutine reads into an N-cube input data, that are problem parameters and given output functions. Input data setup is described in detail in Chapter 4.

Subroutine NCUBE: This subroutine constructs an N-cube according to the number of external variables. This means if the number of external variables is N, this N-cube will contain 2^N vertices. Subroutine NCUBE will examine the input vector in binary form which is assigned to each vertex in the N-cube one by one and link the vertices with the same weight together.

Subroutine CMNL: This subroutine implements Conditional Minimum Labeling in Algorithm DIMN based on the output function values read into the N-cube. As shown in Fig. 3.2.1, this subroutine calls subroutines INCMNT and DSCAN. RF, the minimum number of MOS cells which

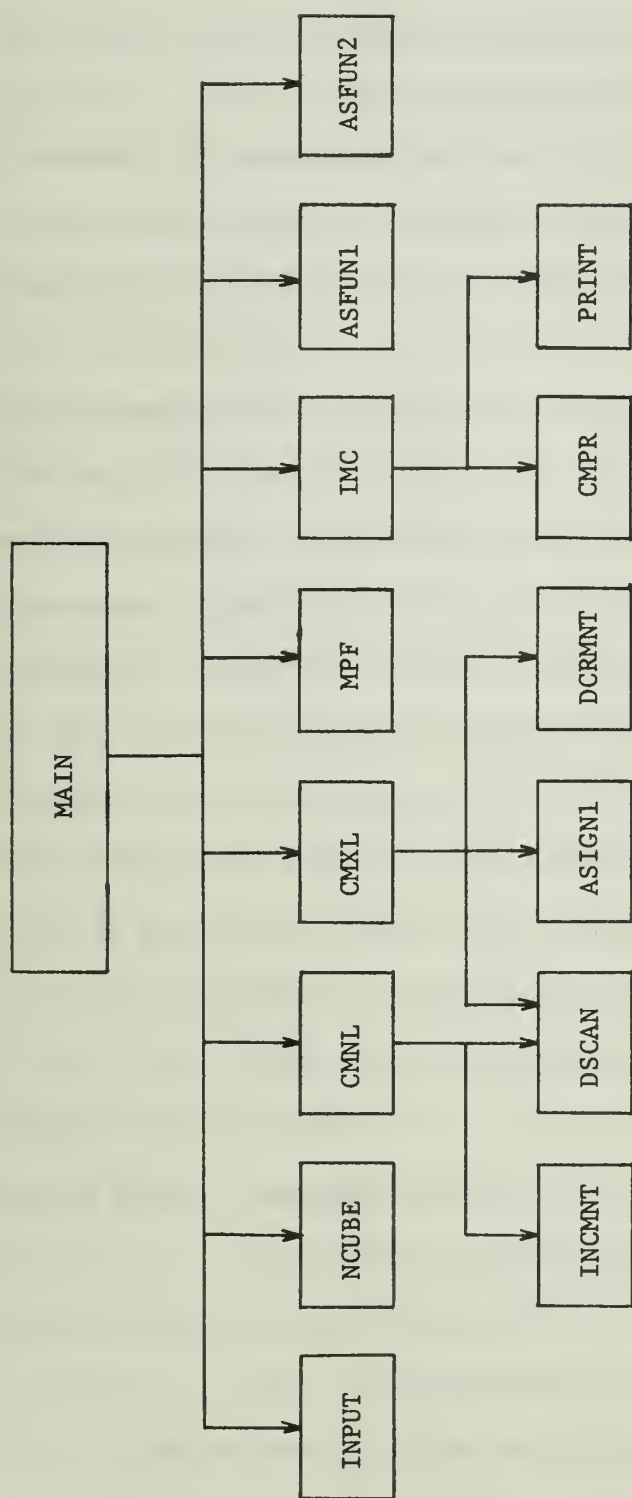


Fig. 3.2.1 General organization of program DIMN.

is required to realize the given function(s) is determined at the first execution of this subroutine CMNL.

Subroutine CMXL: This subroutine implements Conditional Maximum Labeling in Algorithm DIMN based on the output function values read into the N-cube. This subroutine calls subroutine DSCAN, ASIGN1 and DCRMNT.

Subroutine MPF: This subroutine obtains the maximum permissible function from the results obtained in MNL and MXL fields in the N-cube by subroutines CMNL and CMXL, respectively. Then this subroutine stores the resulting MPF into the FUNC field of the Large-cube.

Subroutine IMC: This subroutine obtains an irredundant MOS cell configuration from the maximum permissible function in the Large-cube obtained by subroutine MPF. This subroutine is constructed with the following six steps. This subroutine calls subroutines COMPR and PRINT.

Step 1: In this step, the Large-cube is constructed in the same way as the N-cube is constructed in subroutine NCUBE. The dimension of this Large-cube is determined by the number $N + RF - 1$.

Step 2: This step assigns "0" or "1" to each vertex X which has don't care in the Large-cube such that there exists no vertex Y satisfying the following condition:

Input vector $Y >$ Input vector X and the function value assigned to the vertex Y is "1."

Step 3: This step obtains the set of minimum vectors.

Step 4: This step obtains a subset of the set of minimum vectors which covers all the vertices which originally have the value "0."

Step 5: This step obtains an irredundant subset from the subset obtained in Step 4. The irredundant subset obtained in this step will be used for the realization of an irredundant MOS cell configuration. Since the finding of all irredundant covers (subsets) is essentially the covering problem, it will be very time-consuming as the problem size becomes larger. So in this subroutine, only one irredundant subset is obtained.

Step 6: This step stores values of the function realized by the MOS cell obtained in Step 5 in the LABEL field in the N-cube. We have to be careful that the function value realized at each vertex with don't care by the irredundant MOS cell may be different from the function value assigned to the corresponding vertex in the Large-cube in Step 2.

This step also contains an error checking routine. If the function values realized by the irredundant MOS cell obtained in Step 5 are different from those already stored in the LABEL field of the N-cube, an error message and contents of the array LABEL is printed.

Subroutine ASFUN1: Whenever certain conditions which will be discussed in detail in Section 3.3, are met, Algorithm DIMN does not need to obtain the maximum permissible function. This means that the program loop which consists of CMNL, CMXL and MPF need not be implemented and by detecting such conditions, we can save computation time. This subroutine is called under such circumstances. Then the function values stored in the LABEL field in the N-cube are directly stored by this subroutine in the FUNC field in the Large-cube.

Subroutine ASFUN2: This subroutine is called under a similar condition to that subroutine ASFUN1 is called. The values in the MNL field in the N-cube are stored by this subroutine in FUNC field in the Large-cube.

Subroutine MAIN: This subroutine calls the subroutines described above, whenever necessary, and implements Algorithm DIMN.

3.3 Description About Subroutines

(1) Subroutine MAIN (Fig. 3.3.1)

In block 1, parameters N and M are read in and if these parameters do not satisfy the restriction on problem size (see Section 4.2 for detailed discussion), an error message is printed out and the program execution is terminated. This block also tests EOF (end of file) condition and if all data have been exhausted, the program execution is terminated.

In block 2, the LABEL and DCARE fields in the N-cube are initialized to zero.

In block 3, the subroutine INPUT is called and values on output function cards are read into the LABEL field in the N-cube.

In block 4, the N-cube is constructed based on the parameter value N.

In block 5, II, the pointer which indicates the number of iterations of the program loop is initialized to 1. One MOS cell configuration is determined every time the loop is executed.

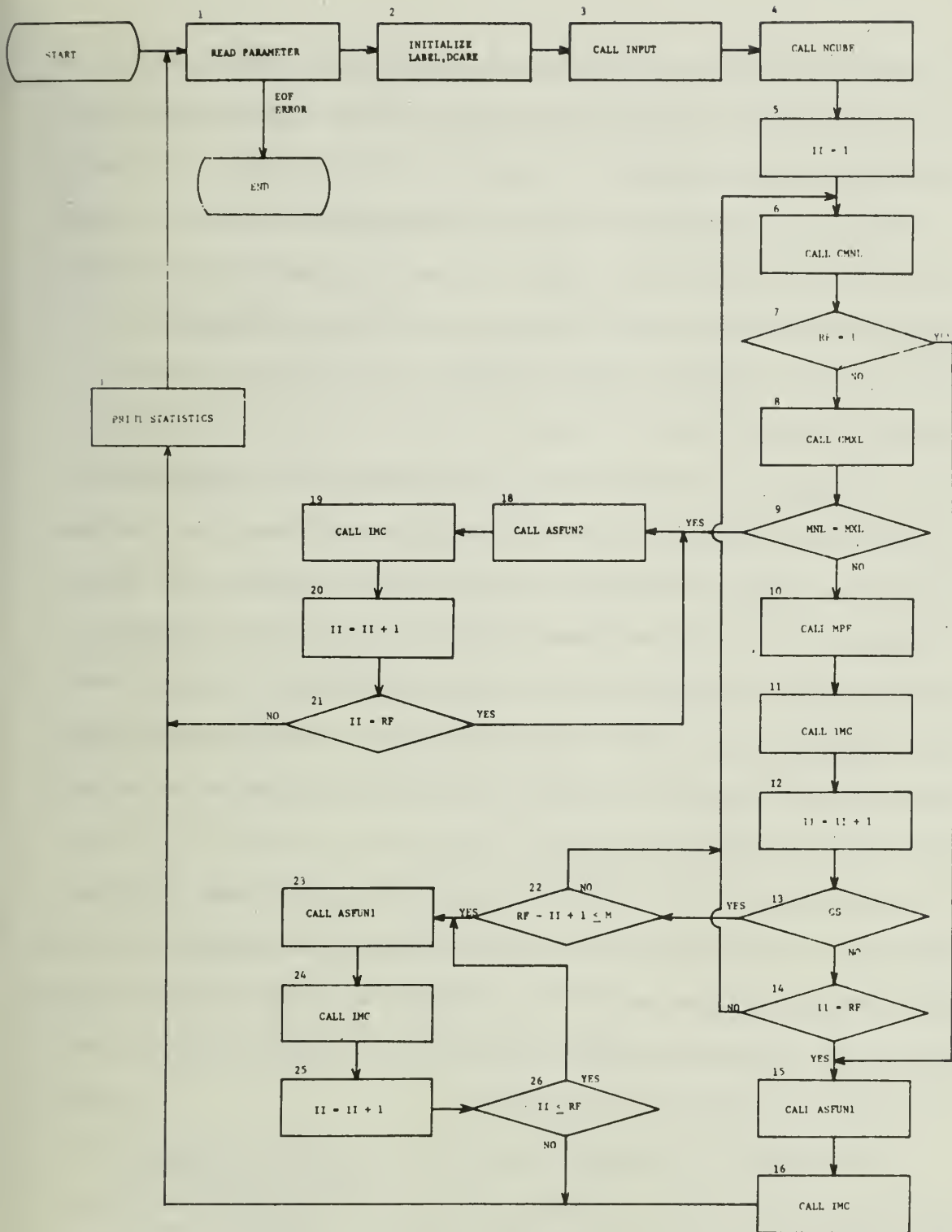


Fig. 3.3.1 Flowchart of subroutine MAIN.

Block 6 implements conditional minimum labeling (CMNL) and at the first execution of this block the value of RF (the minimum number of MOS cells) is determined.

In block 7, the obtained RF value is examined and if it is 1, the maximum permissible function (MPF) need not be obtained; the MOS cell configuration can be obtained immediately by calling ASFUN1 and IMC. If $R \neq 1$, then subroutine CMXL is called to find the conditional maximum labeling.

In block 9, after executing subroutine CMXL, the labels assigned to the N-cube by CMNL and CMXL are compared and if the labels assigned by the two different ways take the same value at every vertex in N-cube, there exists unique minimum negative gate network and block 9 is immediately followed by the loop for obtaining an irredundant MOS cell configuration. Under this circumstance, the sequence for obtaining maximum permissible function, that is, the sequence of subroutines CMNL, CMXL and MPF is skipped.

In block 9, if the labels assigned to the N-cube take different values at some vertices, this block is followed by block 10 (subroutine MPF) and block 11 (subroutine IMC). After the implementation of block 11, the configuration of MOS cell indicated by pointer II is determined (the II-th MOS cell is obtained).

In block 12, pointer II is increased by one and in block 14, if II is not equal to RF, in other words, if we are not going to determine the last MOS cell configuration, the program control is returned to block 6. On the contrary, if II is equal to RF, that is, if the

last MOS cell configuration is going to be determined, the procedure for determining MOS cell configuration is immediately followed (the procedure for obtaining MPF is skipped). After the implementation of block 16, an irredundant MOS network which realizes the given output functions is obtained.

In block 17, the statistics about the obtained network are printed out and the program control is transferred to block 1. The contents of the statistics are described in detail in Chapter 5.

In block 1, if problem cards are not exhausted, a new problem is read in and the entire process is repeated. If problem cards have already been exhausted, the program is terminated.

In block 13, the logical variable CS is examined (this variable is set in subroutine INPUT if all the output functions are completely specified) and if CS is true, this block is followed by block 22. In the case of multiple output functions, the maximum permissible function need not be obtained for the output function which is completely specified. Therefore, if one of the MOS cells which realize the completely specified given output function is being determined, block 22 is immediately followed by the loop for determining MOS cell configuration and the process for obtaining the maximum permissible function is skipped.

(2) Subroutine INPUT

At the beginning of this section, the way how the output functions are stored in arrays LABEL and DCARE will be discussed and then flow-chart will be explained.

As described in Section 3.1, a partially specified negative function sequence is stored in the LABEL and DCARE fields of the N-cube. The internal representation of the LABEL and DCARE fields in the N-cube which has \underline{N} external variables and \underline{M} output functions is shown in Fig. 3.3.2. As shown in this figure, the labels assigned to vertices



u_i - represents the function realized at the output of the i -th MOS cell.

f_i - represents output function f_i .

Fig. 3.3.2 The internal representation of the LABEL and DCARE fields in the N-cube.

in the N-cube are stored in bit pattern. The column labeled with u_i stores the function realized at the output of the i -th MOS cell and

the column labeled with f_1 stores the given output function f_1 . As will be discussed in Chapter 4, since the output functions are supplied in truth table form, output function 1 is stored in column f_1 , output function 2 is stored in column f_2 and so forth. In general, output function i is stored in column f_i .

The function value of the i -th function is stored in the LABEL and DCARE fields of vertex $(j-1)$ in the following way. (As described in Section 3.1, the LABEL and DCARE fields of vertex $(j-1)$ are stored in LABEL (j) and DCARE (j) , respectively.)

If a function value is zero, zero is stored by assigning zero to column f_1 of both LABEL (j) and DCARE (j) . (Since arrays LABEL and DCARE have already been initialized to zero in block 2 of subroutine MAIN, no action need be taken.) If a function value is one, one is stored by assigning one to column f_1 of LABEL (j) and zero to column f_1 of DCARE (j) . This is implemented by adding 2^{M-1} to LABEL (j) . Finally, if a function value to be stored is don't care, this is done by assigning zero to column f_1 of LABEL (j) and one to column f_1 of DCARE (j) . This is implemented by adding 2^{M-1} to DCARE (j) . The example in Fig. 3.3.3 shows how the output functions in the right hand side are stored in arrays LABEL and DCARE for $N=3$ and $M=2$. In the truth table of this figure, * represents don't care condition.

The following is the definition of each variable which appears in subroutine INPUT.

NCARD: This stores the number of input data cards required to describe one output function.

	LABEL		DCARE						
	f_1	f_2	f_1	f_2	x_1	x_2	x_3	f_1	f_2
(1)		1	0		0	0	0	1	*
(2)		0	0		0	0	1	*	0
(3)		0	1		0	1	0	0	1
(4)		1	0		0	1	1	1	*
(5)		0	0		1	0	0	*	0
(6)		0	0		1	0	1	*	*
(7)		0	1		1	1	0	0	1
(8)		1	1		1	1	1	1	1

Fig. 3.3.3 Example: how output functions are stored in arrays LABEL and DCARE.

I: This is an indicator of output function number. If problem requires M output functions, I changes from one to M.

J: This is an indicator of card number within one output function. J changes from one to NCARD.

K: This is a column indicator in one card. Therefore, K changes from one to 80.

VTEX: This is a pointer to a vertex in the N-cube in which an output function value is to be stored. When VTEX = i, this points to vertex (i-1).

CHAR (80): This array is a character buffer for input data. This implies that one output function card is read in at a time.

CS: This is a logical variable which is set when every output function is completely specified.

Fig. 3.3.4 shows the flowchart of subroutine INPUT. In block 1, CS is set. If don't care is found in output functions, it will be reset in block 9. This means that after executing subroutine INPUT, CS is set if and only if every output function is completely specified.

In block 2, NCARD is obtained in the following way. If 2^N is devisable by 80, then $NCARD = 2^N/80$, otherwise $NCARD = 2^N/80+1$, where / represents integer division and the result of $2^N/80$ is equal to $\lfloor \frac{2^N}{80} \rfloor$.

In block 7, a function value which is stored in CHAR(K) is examined and the function value is stored in the N-cube in the way described in the beginning of this section. In the input format, a blank means the termination of the description of one output function. Therefore, if a character in CHAR(K) is a blank, this is interpreted as the termination of the description of one output function and, in block 15, an error is checked. In a normal termination of the description of one output function, at the instant when a blank character has been received, J and VTEX have to take the following value;

$$J = NCARD, \quad VTEX = 2^N + 1$$

If an error occurs, an error message is printed and the control returns to block 1 in subroutine MAIN. This means that this problem has been skipped. In block 7, receiving characters other than zero, one, * and blank is also interpreted as an error.

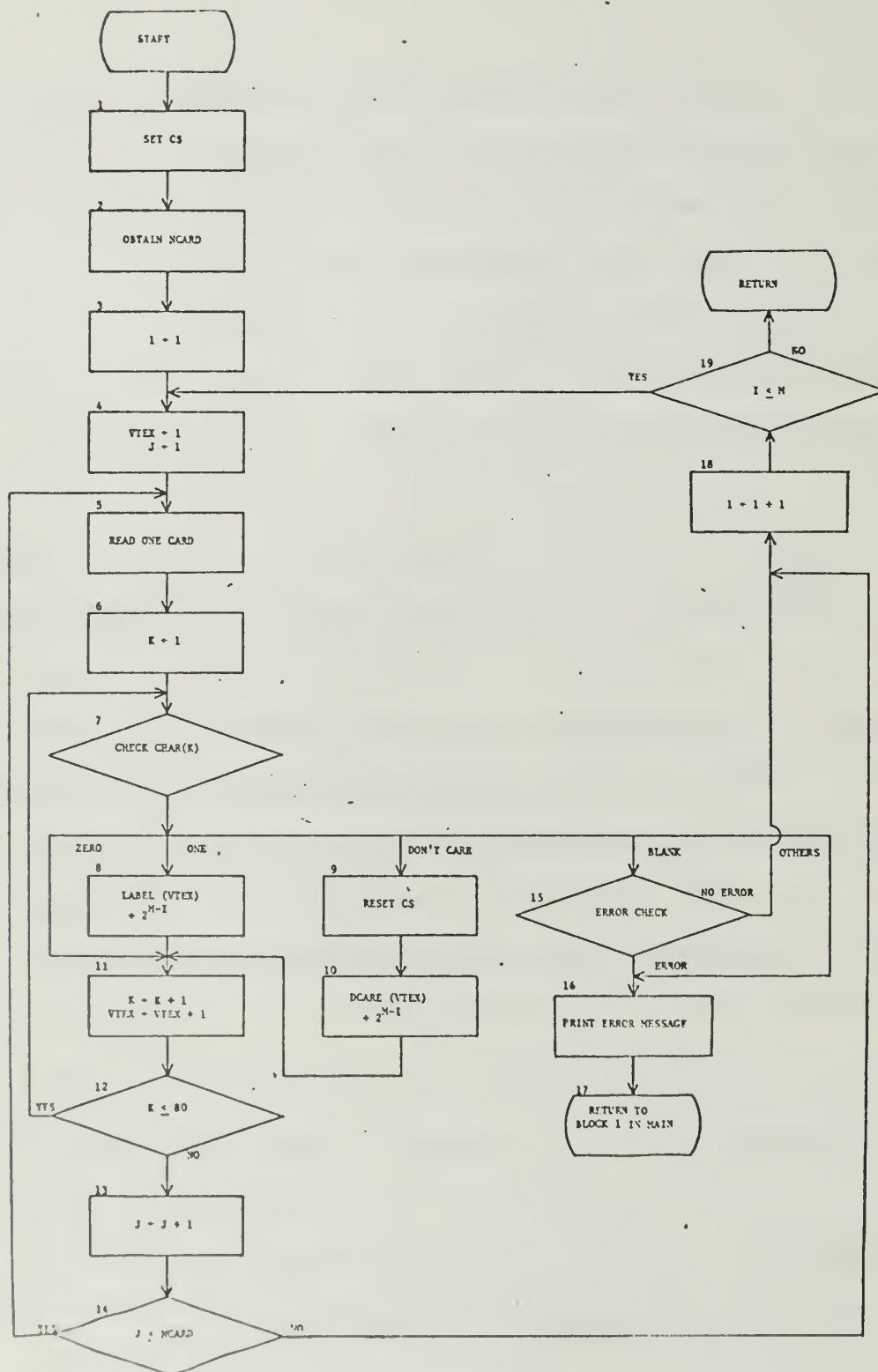


Fig. 3.3.4 Flowchart of subroutine INPUT.

(3) Subroutine NCUBE

Flowchart is shown in Fig. 3.3.5. The following is the definitions of variables which appear in this subroutine.

I: This stores an input vector in binary form.

X: This is a variable in which each input vector is examined bit by bit and the weight of the input vector is determined.

W: This is a variable in which the weight of the input vector is stored.

STARTL, ENDL and other arrays are described in Section 3.1.

When the number of external variables is N , there exists 2^N input vectors (ranging from 0 to $2^N - 1$ in binary form). Therefore, the procedure which determines the weight of an input vector and links the input vectors with the same weight is repeated for $I=0$ to $2^N - 1$. In the loop which consists of blocks 4, 5, 6 and 7, each input vector is examined bit by bit and in blocks 8, 9 and 10, the input vectors with the same weight are linked together. For this chain constructing purpose, STARTL has to be initialized to zero in block 1. An example of the N -cube constructed in this procedure is shown in Fig. 3.1.5 for $N=3$.

(4) Subroutine CMNL

Flowchart is shown in Fig. 3.3.6. The following is the definition of variables which appear in this subroutine.

USPFY: As previously mentioned, array LABEL stores a partially specified negative function sequence. USPFY stores the number of unspecified bits in array LABEL. The relation between USPFY, RF, II and M is shown in Fig. 3.3.7.

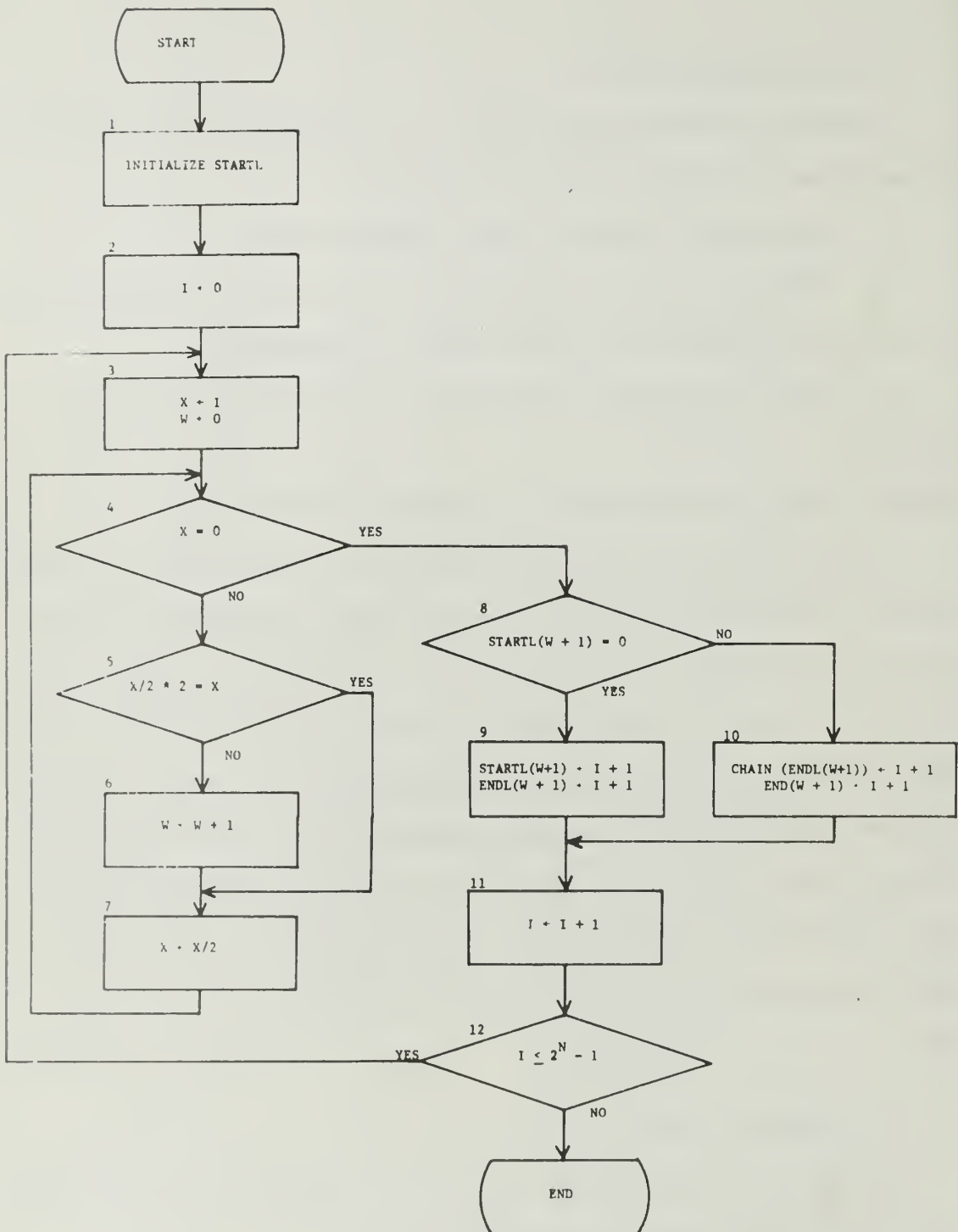


Fig. 3.3.5 Flowchart of subroutine NCUBE.

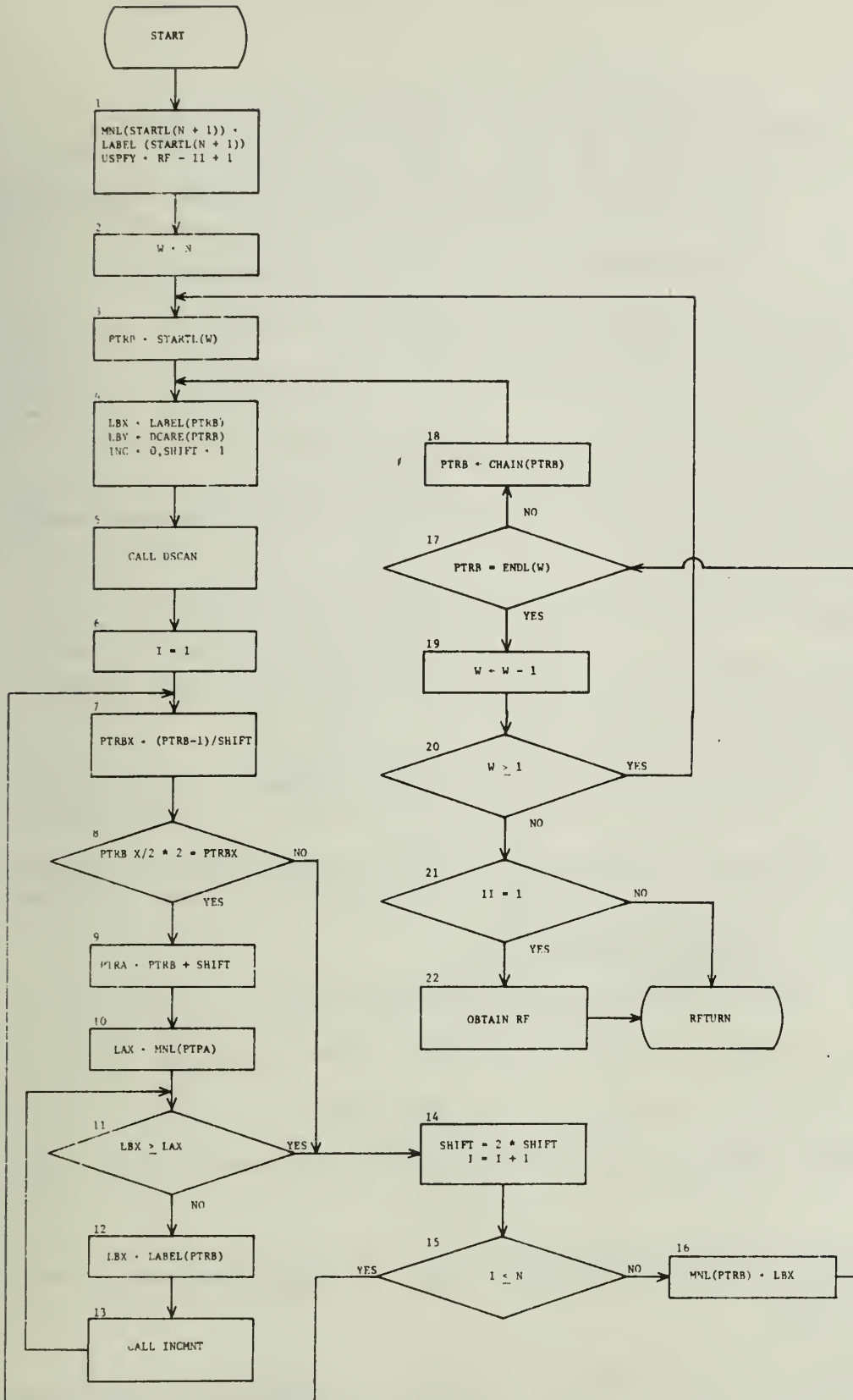


Fig. 3.3.6 Flowchart of subroutine CMNL.

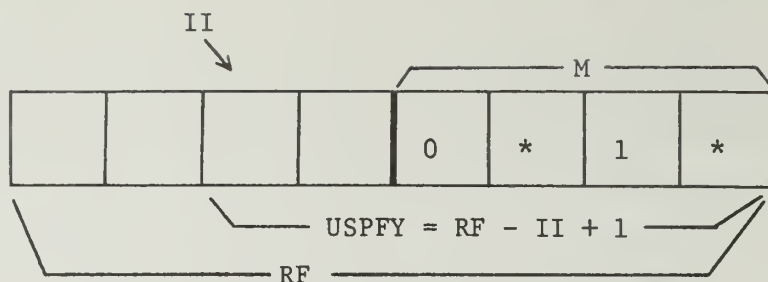


Fig. 3.3.7 An element of array LABEL.

W: This stores an actual weight plus one.

PTRB: This is a pointer to vertex B to which the minimum possible value is to be assigned.

PTRA: This is a pointer to vertex A which is connected to vertex B by the edge from vertex A to vertex B. The vertex A has a greater weight than vertex B by one.

LBX: This is a variable in which LABEL pointed by pointer B is increased.

LAX: This is a variable which stores the minimum label which has already been assigned to vertex A.

LBX: This is a variable in which DCARE pointed by pointer B is examined in subroutine DSCAN. The number of don't care bits and the weight assigned to each don't bit are determined.

NDCARE: This is a variable which stores the number of don't care bits.

BWEIT: This is an array which stores the weight assigned to each don't care bit.

PTRBX: This is a variable in which the input vector in binary form assigned to vertex B is examined bit by bit. This variable, together with variable SHIFT, determine PTRB by calculation.

INC: This is a counter for the increment of LBX. This is described in detail in subroutine INCMNT.

In block 1 (Fig. 3.3.6), the minimum possible label is assigned to a vertex with weight N. Since LABEL is initialized to zero in block 2 in subroutine MAIN (Fig. 3.3.1), the unspecified bits in LABEL (STARTL (N+1)) have already been assigned zero, the minimum possible value. As shown in Fig. 3.3.7, the number of unspecified bits is obtained by calculating $RF - II + 1$. As RF is obtained after the first implementation of subroutine CMNL, the value of RF has to be specified in subroutine MAIN before CMNL is called for the first time.

After assigning a label to the vertex with weight N, a minimum possible label is assigned to every other vertex in the following way. In blocks 4, 5, ..., 16, each vertex pointed by PTRB (vertex B) is assigned a minimum possible label. In general, as shown in Fig. 3.3.8,

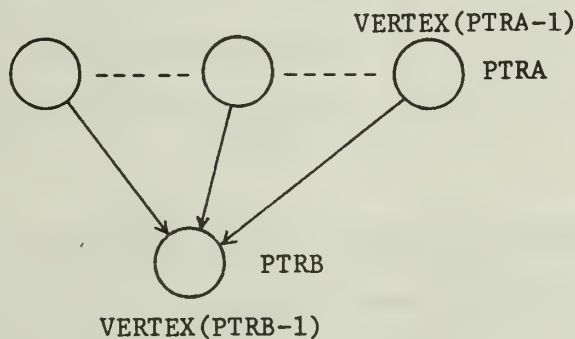


Fig. 3.3.8 Relation between vertices A and B in CMNL.

there exist several vertices which are directly connected to vertex B by the edges from these vertices to vertex B. Minimum labels which have already been assigned to these vertices are compared with LBX

one by one (block 11, LABEL (PTRB) is assigned to LBX in block 4) and LBX is increased by one (block 13) until the condition that no one of the labels assigned to these vertices is larger than the label assigned to vertex B is satisfied. In block 9, PTRB is calculated by $PTRA = PTRB + SHIFT$. As vertices with the same weight, say (w-1), are linked by CHAIN field, starting from the vertex pointed by STARTL(W), all the vertices within the linked list can be exhausted by utilizing this scheme (blocks 3, 17 and 18).

Starting from the vertices with weight (N-1), the above procedure is applied to every group of vertices with the same weight until a label is assigned to the vertex with weight zero (blocks 2, 19, 20). At the first implementation of this subroutine, RF value is obtained (blocks 21, 22).

(5) Subroutine DSCAN

Flowchart is shown in Fig. 3.3.9. This subroutine has four parameters; two of them are input parameters (LBY, USPFY) and the others are output parameters (NDCARE, BWEIT). It is called by CMNL and CMXL.

It determines the number of don't care bits in the function part of the label assigned to vertex B and the weight assigned to these don't care bits. This is accomplished by examining LBY (don't care field of vertex B) bit by bit. If "1" is encountered, NDCARE is increased by one and the corresponding bit weight is stored in BWEIT (NDCARE). This bit weight is represented in variable SHIFT (this stores the value 2^{I-1}). The above procedure is continued until all the unspecified bits are examined.

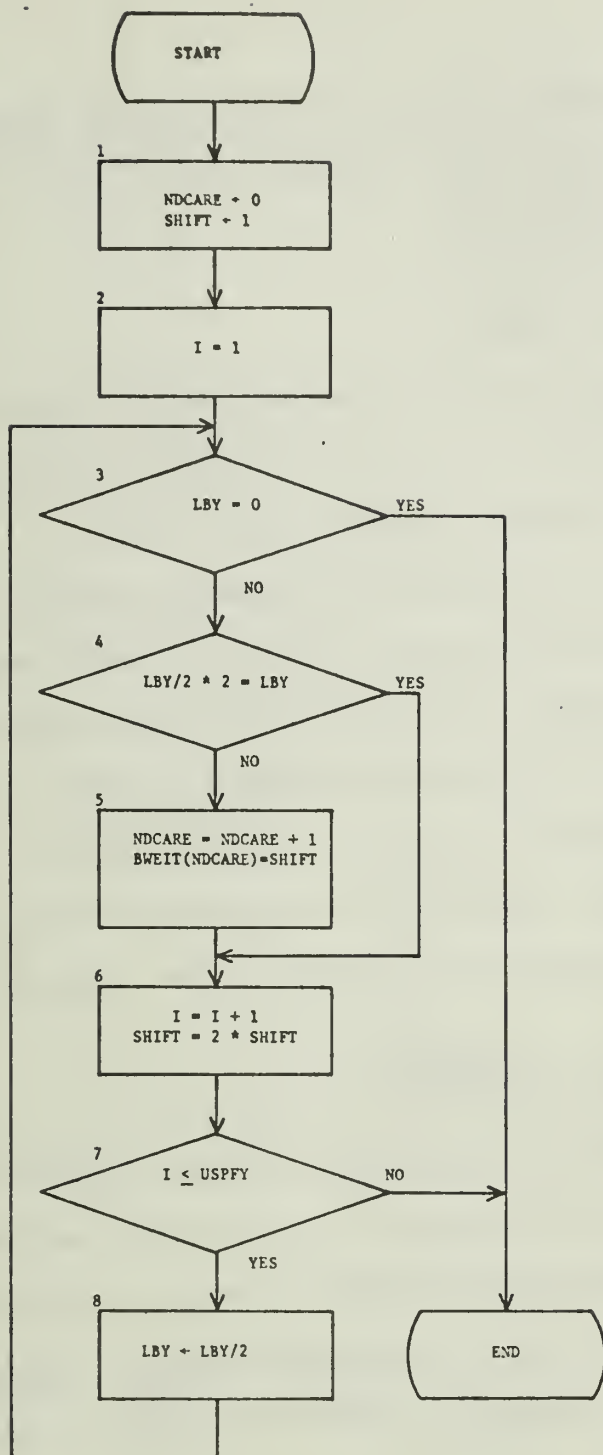


Fig. 3.3.9 Flowchart of subroutine DSCAN.

An example is shown in Fig. 3.3.10 for a label assigned to vertex B. The label in this example contains 2 don't care bits and the weight of these bits are 2^0 and 2^2 .

(6) Subroutine INCMNT

This subroutine has five parameters; three of them (NDCARE, BWEIT, M) are input parameters and the others (INC, LBX) are input-output parameters. The flowchart is shown in Fig. 3.3.11.

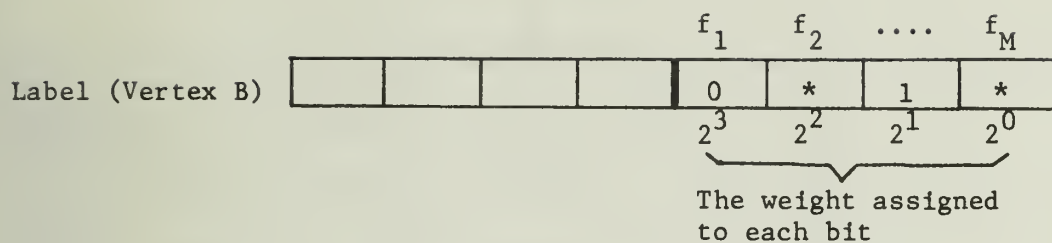
Although INC is a counter for incrementing LBX (the label field of vertex B), each bit in counter INC has a different weight from an ordinary binary counter. Bit weights of the least significant half which correspond to the bit weights of don't care bits in LBX can be obtained in array BWEIT by subroutine DSCAN (the number of these bits is stored in NDCARE). Weights of other unspecified bits can be obtained by the following formula; the bit weight of the J-th least significant bit is

$$2^{M + J - \text{NDCARE} - 1}$$

where J is the index which indicates the number of program loop iterations in Fig. 3.3.11. An example of the bit weight assignment to counter INC is shown in Fig. 3.3.10 for a label of vertex B.

In block 1 (Fig. 3.3.11), counter INC which has already been initialized to zero in block 4 in Fig. 3.3.6 (CMNL) is increased by one.

In block 3, variable SHIFT and index J are initialized so that SHIFT stores the bit weight of the J-th least significant bit. In the loop which consists of blocks 4,...,10, the contents of counter



NDCARE

2

BWEIT(1)

2^0

BWEIT(2)

2^2

BWEIT(3)

--

BWEIT(4)

--

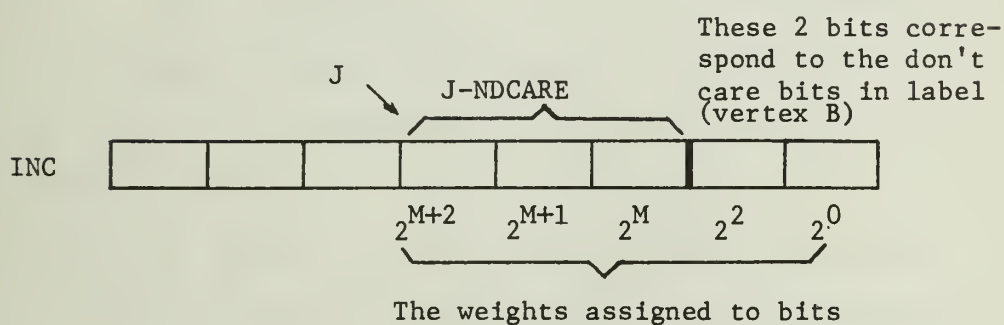


Fig. 3.3.10 Example: how don't care bits in the LABEL field is treated.

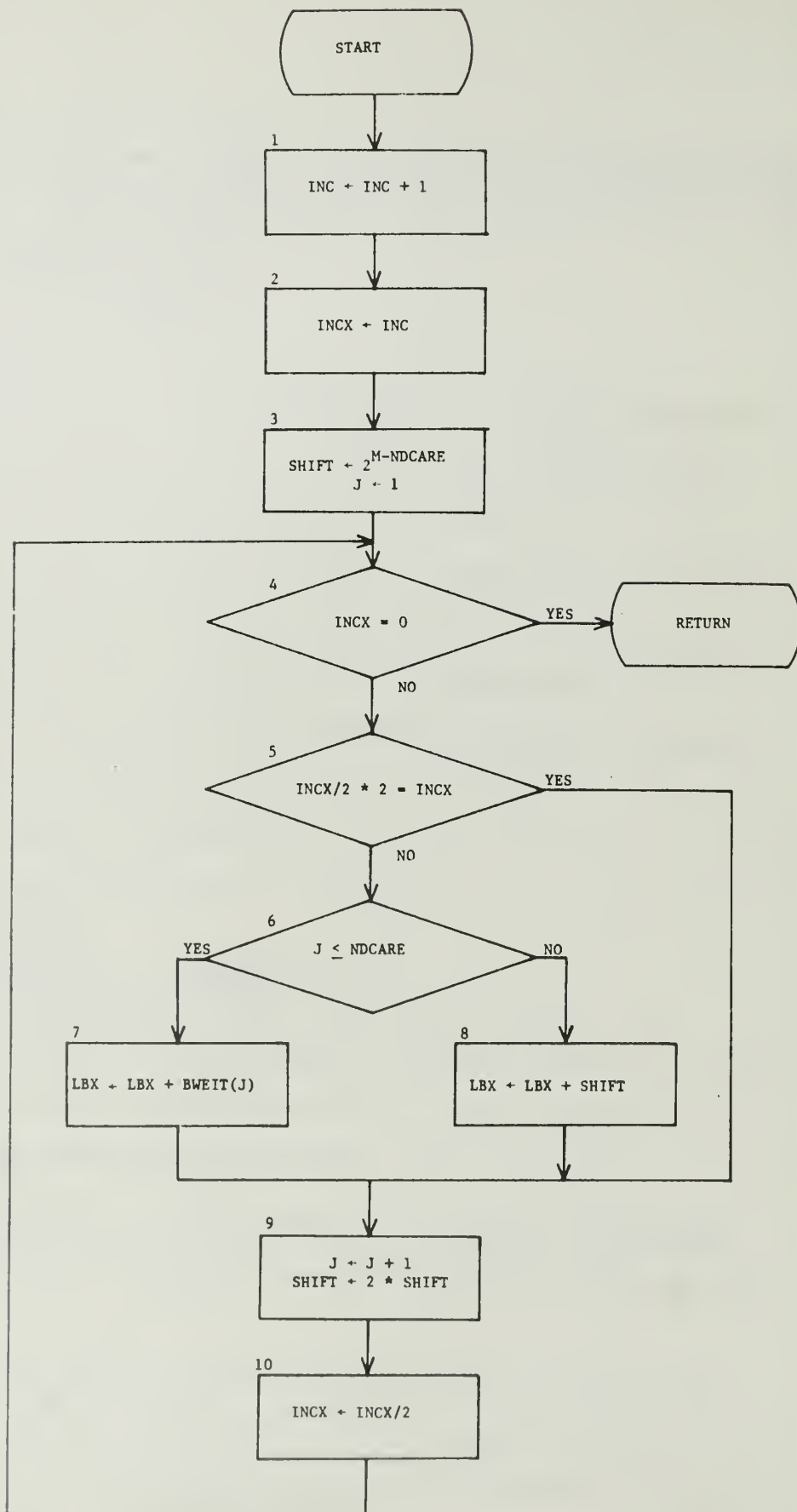


Fig. 3.3.11 Flowchart of subroutine INCMNT.

INC are examined bit by bit and if "1" is encountered, the weight assigned to this bit of 1 is added to LBX.

In block 6, the value of index J is examined and if J is not greater than NDCARE, the contents of BWEIT(J) is added to LBX, and otherwise the contents of the variable SHIFT is added.

After executing this subroutine, LBX is increased to the next possible state and then compared with LAX in block 11 of subroutine CMNL (Fig. 3.3.6). It is to be noted that LBX is not always increased by one because it may contain already specified bits. As shown in block 12 (Fig. 3.3.6), prior to calling this subroutine, LBX has to be initialized to the value stored in LABEL (PTRB).

(7) Subroutine CMXL

This subroutine is executed in almost the same way as the subroutine CMNL is. The only difference is that after 1's are assigned to the unspecified bits of label field of vertex B by calling subroutine ASIGN1, the label is decreased instead of being increased in subroutine CMNL.

The flowchart is shown in Fig. 3.3.12 and the following is the definitions of variables used in this subroutine. Variables with the same definition as those in subroutine CMNL are omitted.

PTRB: This is a pointer to vertex B to which the maximum possible value is to be assigned.

PTRA: This is a pointer to vertex A which is connected to vertex B by the edge from vertex B to vertex A. The vertex A has a smaller weight than vertex B by one. The relation between vertex A and vertex B is shown in Fig. 3.3.13.

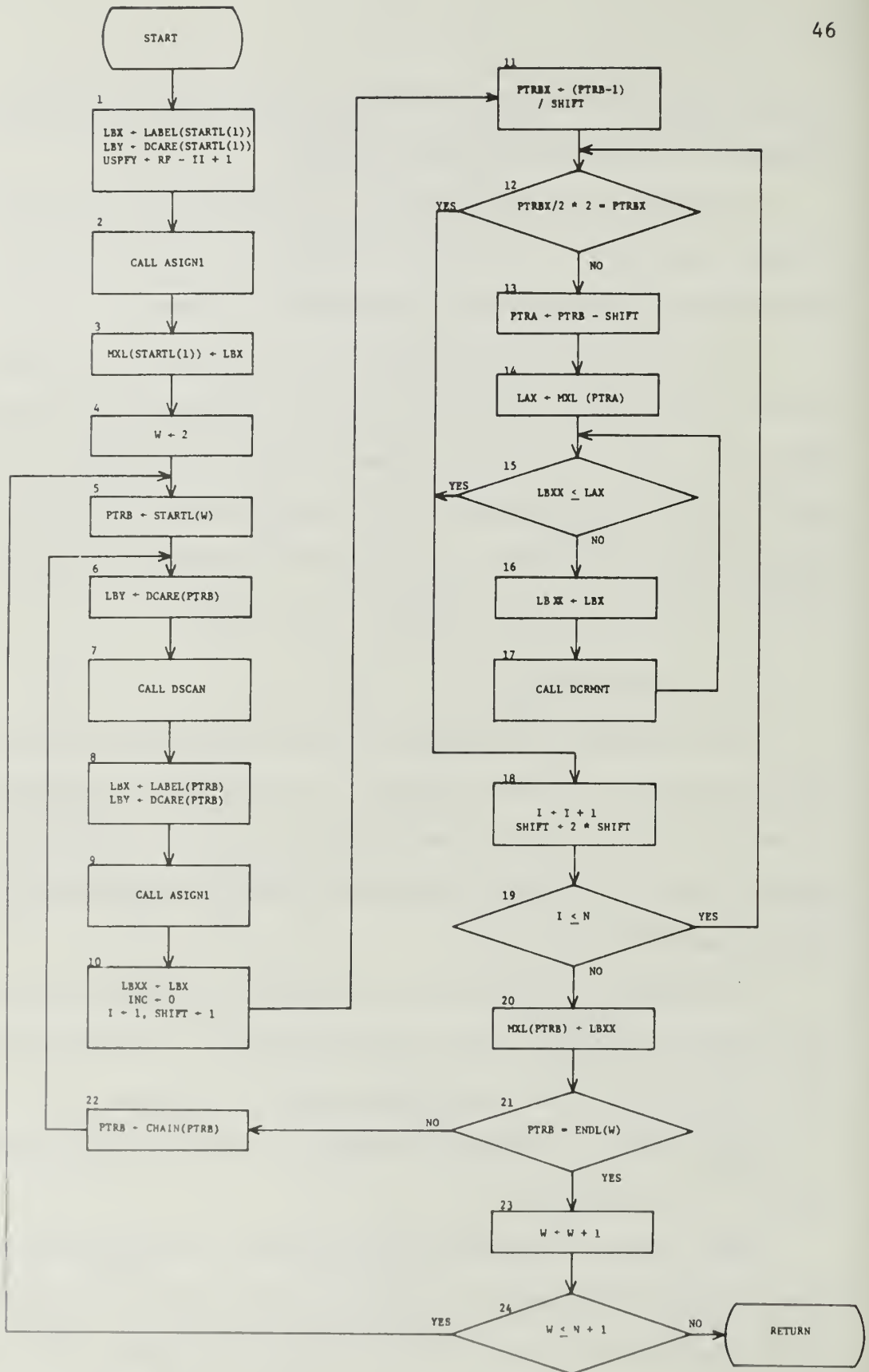


Fig. 3.3.12 Flowchart of subroutine CMXL.

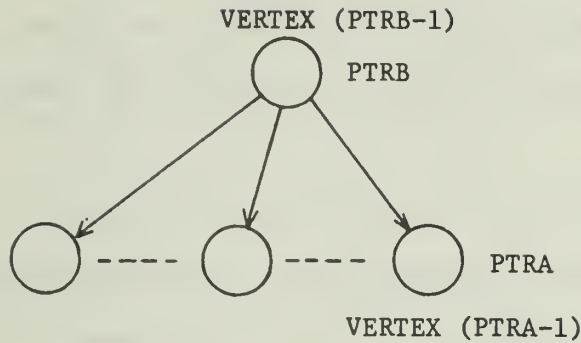


Fig. 3.3.13 Relation between vertices A and B in CMXL.

LBX: This is a variable where LABEL (PTRB) is stored after 1's are assigned to the unspecified bits in LABEL (PTRB).

LBXX: This is a variable in which contents of LBX is determined.

LAX: This is a variable which stores the maximum label which has already been assigned to vertex A.

INC: This is a counter for decreasing LBXX. The contents of INC is subtracted from LBXX in subroutine DCRMNT.

In blocks 1, 2 and 3 (Fig. 3.3.12), the maximum possible label is assigned to the vertex with weight zero. Starting from the vertices with weight one (block 2), a maximum possible label is assigned to every vertex in the N-cube in a similar manner as in subroutine CMNL. In block 13, PTR A is obtained by subtracting SHIFT from PTRB.

(8) Subroutine ASIGN1

This subroutine has four parameters; three of them (USPFY, M and LBY) are input parameters and the remaining one (LBX) is an input-output parameter. One is assigned to every unspecified bit of LBX

including unspecified don't care bits in the values of output functions assigned to vertex B. This is accomplished by examining input parameter LBY.

In Fig. 3.3.14, I is a variable which points to a bit in the label field of vertex B. Starting from the least significant bit of the label ($I=1$), one is assigned to all the unspecified bits by increasing the index I. SHIFT is a variable which represents the weight assigned to each unspecified bit indicated by index I.

In block 2, the value of USPFY is examined and if it is larger than M (the number of output functions), then in the loop which consists of blocks 3, 4, 5, 6 and 7, the values of output functions assigned to vertex B (M bits) is examined. In this loop, each time when a don't care bit is encountered, one is assigned to the don't care bit by adding SHIFT to LBX. Then, in the loop which consists of blocks 8, 9 and 10, one is assigned to every remaining unspecified bit. If the value of USPFY does not exceed M, then the unspecified bits in the label field of vertex B, the number of which is specified in input parameter USPFY, are examined and one is assigned to every unspecified don't care bit.

(9) Subroutine DCRMNT

This subroutine has five parameters; three of them (NDCARE, BWEIT and M) are input parameters and the others (INC, LBXX) are input-output parameters. In this subroutine, parameter LBXX is decreased to the next possible state. As shown in block 16 in Fig. 3.3.12 (CMXL), prior to calling this subroutine, LBXX has to be initialized to the value stored in LBX.

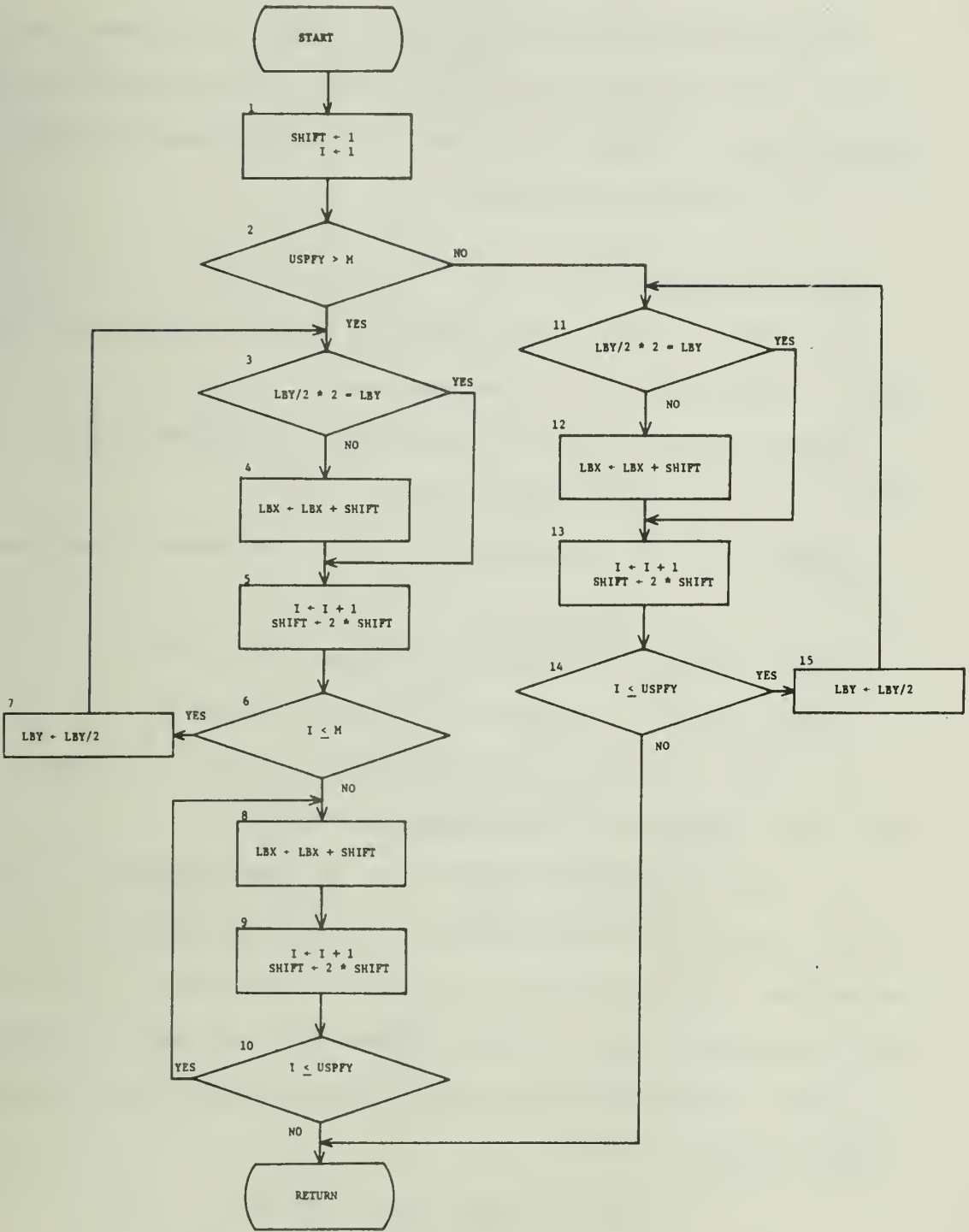


Fig. 3.3.14 Flowchart of subroutine ASIGN1.

As can be seen in the flowchart (Fig. 3.3.15), this subroutine is executed in the same way as subroutine INCMNT. The only exception is blocks 7 and 8. In these blocks, BWEIT(J) and SHIFT are subtracted from LBXX instead of being added.

(10) Subroutine MPF

Flowchart is shown in Fig. 3.3.16 and the following is the definition of variables used in this subroutine.

MXVTXL: This is a variable which stores the maximum value, $(2^N - 1)$, of vertex number in the N-cube.

NVTEXF: This is a variable which stores the number of vertices, (2^{N+II-1}) , in the Large-cube.

RSHIFT: This is a variable which stores value 2^{RF-II} .

LSHIFT: This is a variable which stores value 2^{II-1} .

I: This is an index of the program loop in Fig. 3.3.16 and at the same time I represents a vertex number in the N-cube.

J: This is a variable which stores the vertex number of a vertex in the Large-cube to which the maximum permissible function value is to be assigned. As can be seen in Fig. 3.3.16, when vertex I in the N-cube is concerned, the corresponding II-th MPF value will be assigned to vertex J in the Large-cube, where J is calculated by the following formula (see Fig. 3.3.17);

$$\begin{aligned}
 J &= I * 2^{II-1} + \text{LABEL } (I+1) / 2^{RF-II+1} \\
 &= I * \text{LSHIFT} + \text{MNL } (I+1) / 2 * 2^{RF-II} \\
 &= I * \text{LSHIFT} + \text{MNLX} / 2
 \end{aligned}$$

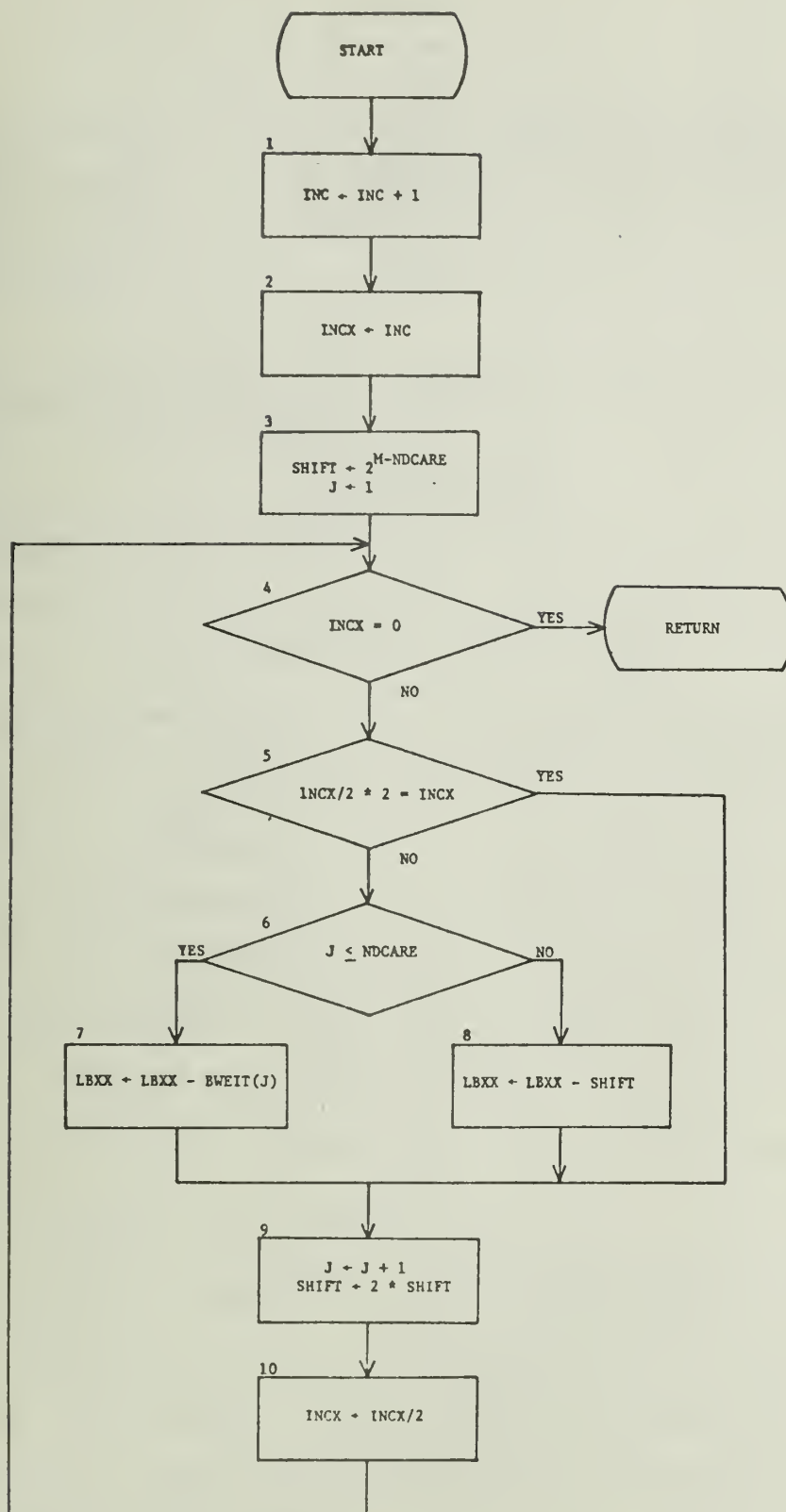


Fig. 3.3.15 Flowchart of subroutine DCRMNT.

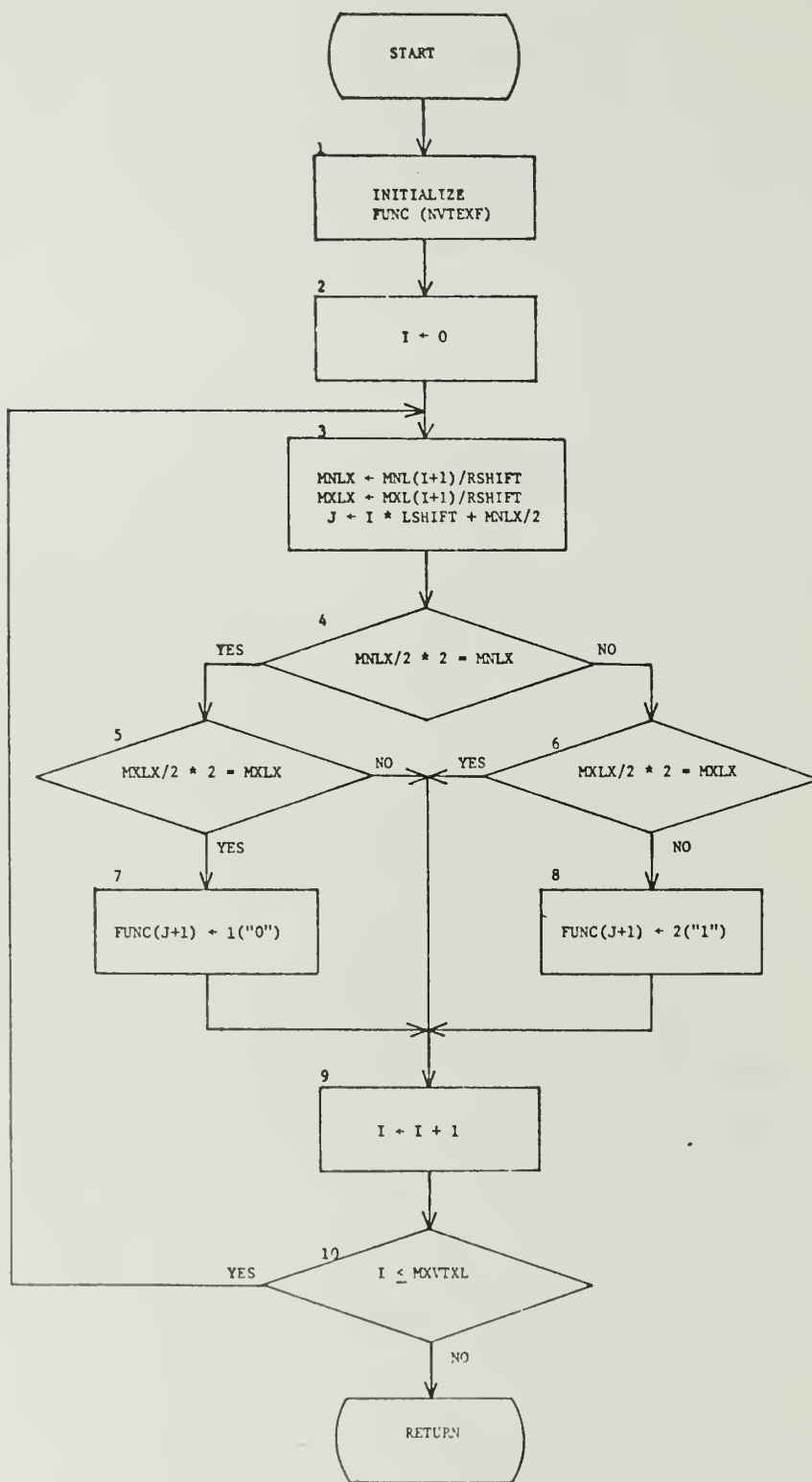


Fig. 3.3.16 Flowchart of subroutine MPF.

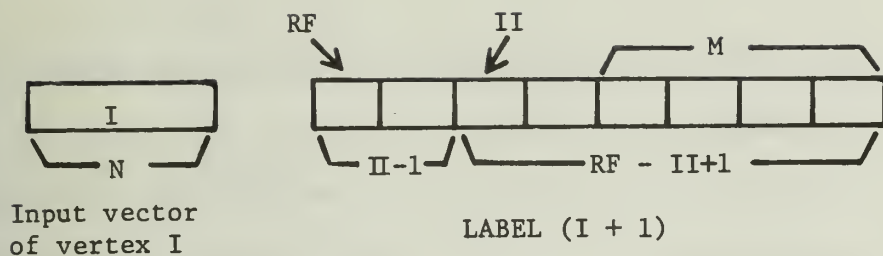


Fig. 3.3.17 Input vector and LABEL of vertex I in the N-cube.

In block 1 (Fig. 3.3.16), the FUNC field of the Large-cube is initialized to zero. In the loop which consists of blocks 3, 4, ..., 10, the II -th most significant bits of MNL ($I+1$) and MXL ($I+1$) are compared. If the resulting MPF value is "0," 1 is assigned to FUNC ($J+1$), if the MPF value is "1," 2 is assigned to FUNC ($J+1$) and if the MPF value is "don't care," 0 is assigned to FUNC ($J+1$). (In the last case nothing should be done because FUNC is already initialized to zero in block 1.) We should notice that MNL ($I+1$) and MXL ($I+1$) store the minimum and the maximum labels assigned to vertex I in the N-cube, respectively, and FUNC ($J+1$) stores the FUNC field of vertex J in the Large-cube. This loop is repeated until all the vertices in the N-cube are exhausted.

(11) Subroutine IMC

As described in Section 3.2, this subroutine obtains an irredundant MOS cell configuration from the maximum permissible function in the Large-cube. This subroutine consists of six steps which will be described below. Before describing each step in detail, the definitions of variables which appear in these steps in common are explained first.

(The definitions of variables MXVTXL, NVTEFX, LSHIFT and RSHIFT already appeared in the description of subroutine MPF.)

INPD: This is a variable which stores the dimension of the Large-cube which is calculated in $N + II - 1$. This variable also represents the dimension of input vectors in the Large-cube.

NVTEXL: This is a variable which stores the number, 2^N , of vertices in the N-cube.

MXVTXF: This is a variable which stores the maximum value, $2^{N+II-1} - 1$, of vertex number in the Large-cube.

ERROR: This is a logical variable which stores an error status detected in Step 6 of subroutine IMC.

Step 1: This step constructs a Large-cube in the same way as subroutine NCUBE constructs a N-cube. The flowchart is shown in Fig. 3.3.18. In this flowchart variable Y corresponds to variable X in subroutine NCUBE. Other variables have the same definitions as in subroutine NCUBE.

Step 2: The flowchart for step 2 is shown in Fig. 3.3.19. In block 1, the FUNC field of the vertex with weight INPD is examined and if it contains don't care, "0" is assigned to the vertex. Then all the vertices in the Large-cube are traversed in the same way as in subroutine CMNL and every vertex which was assigned don't care in subroutine MPF is assigned "0" or "1" in the following way.

As in subroutine CMNL, vertex B to which value "0" or "1" is going to be assigned is pointed by PTRB. The values assigned to all the

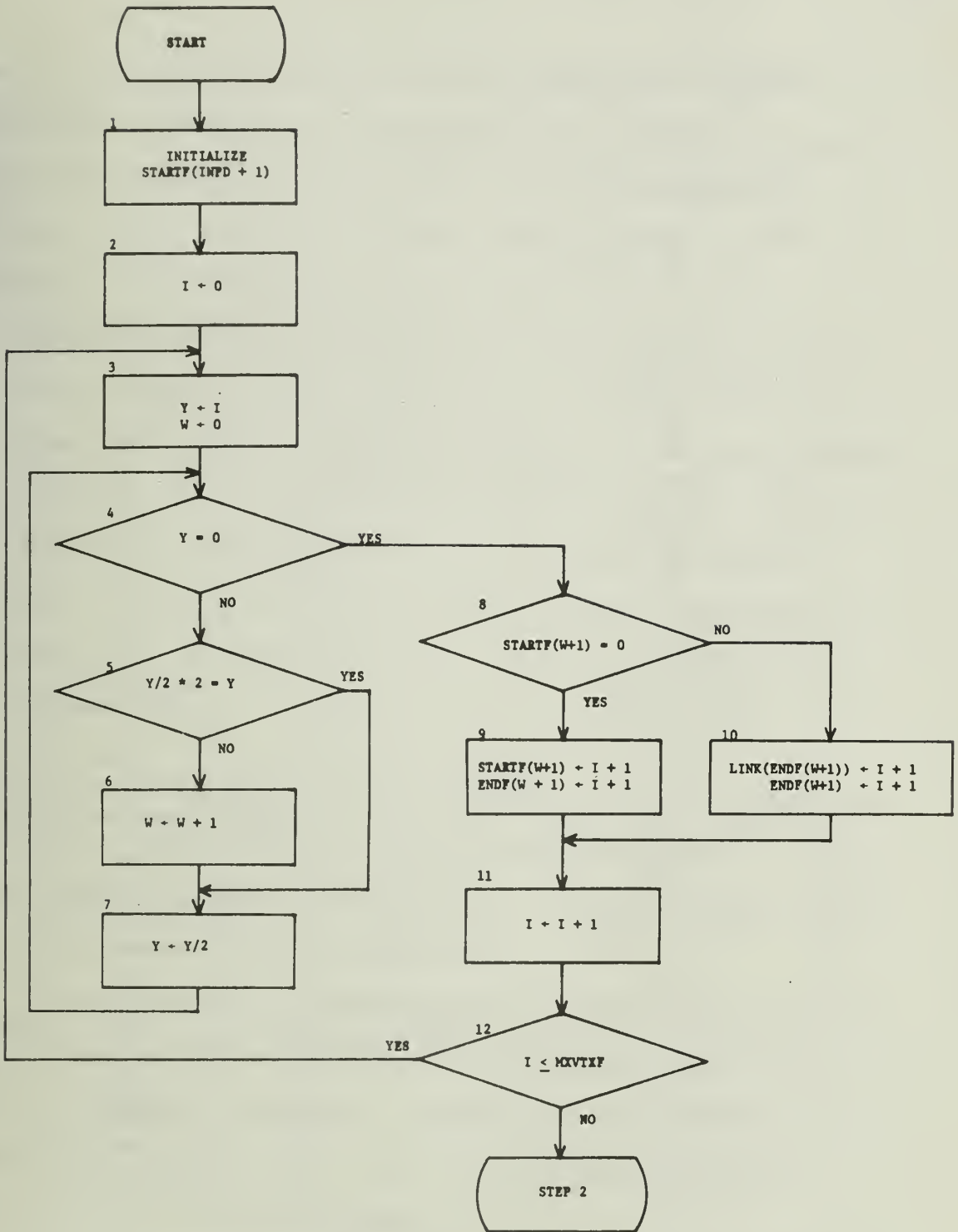


Fig. 3.3.18 Flowchart of subroutine IMC. (Step 1)

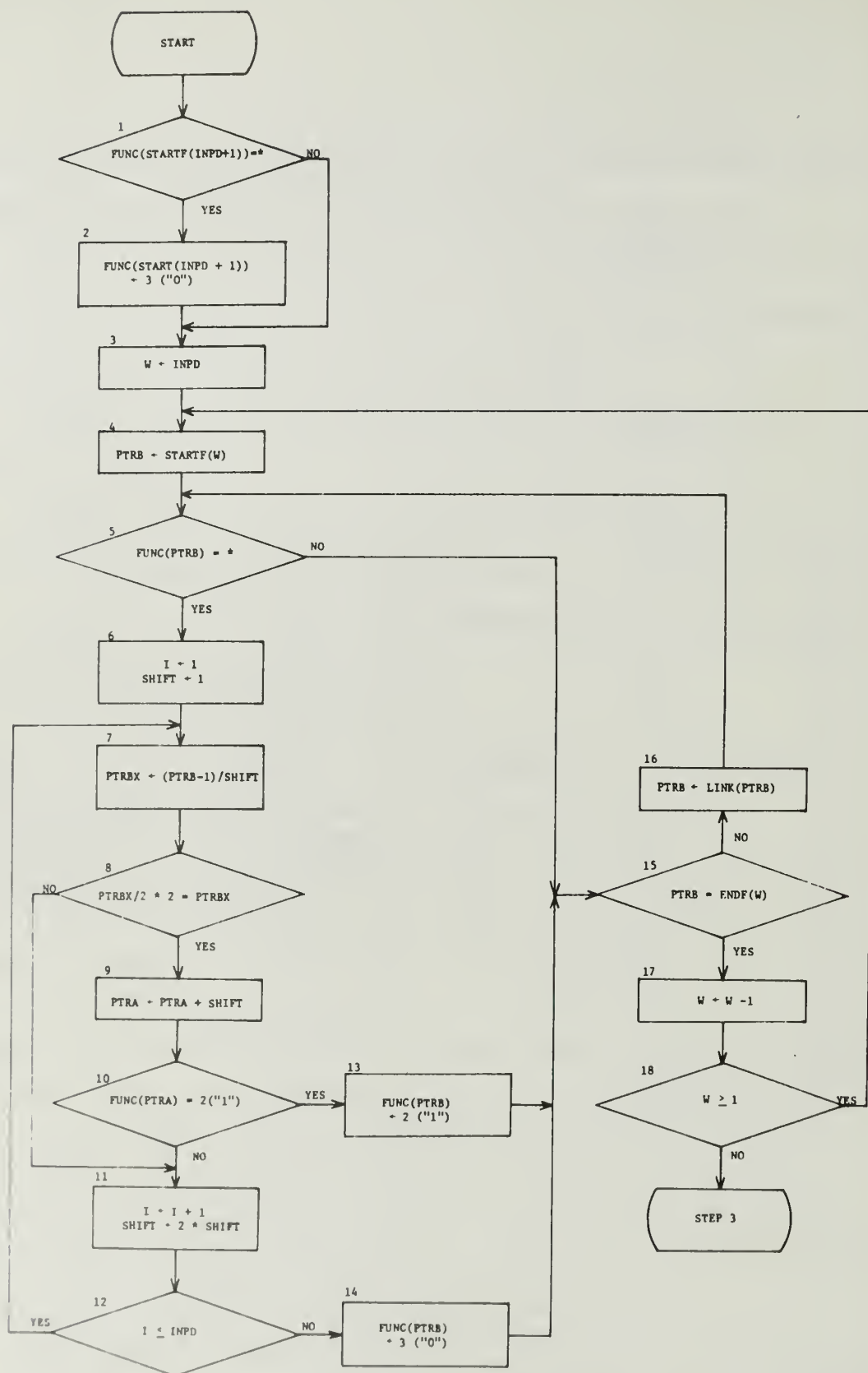


Fig. 3.3.19 Flowchart of subroutine IMC. (Step 2)

vertices which are connected to vertex B by the edges from these vertices are examined in the loop which consists of blocks 7,...,12. If all these vertices take the value "0" (1 or 3), then vertex B is assigned "0" by storing 3 to the FUNC field. Otherwise vertex B is assigned "1" by storing 2 to the FUNC field.

In order to get an irredundant cover, it is necessary to distinguish the original "0" which has already been assigned in subroutine MPF from the "0" which is assigned in this step. This is accomplished by assigning 1 or 3 to the FUNC field of the Large-cube according to the type of "0" (1 for original "0" and "3" for the "0" which is assigned in this step). In the following flowcharts, the original "0" is expressed as "0*" and the "0" which is assigned in this step is expressed as "0."

Step 3: This step obtains the set of minimum vectors. The flowchart for step 3 is shown in Fig. 3.3.20. In this step, NMINV is a variable which stores the number of minimum vectors and MINV is an array which stores the minimum vectors in binary form.

All the vertices in the Large-cube are traversed in the same way as in subroutine CMNL and every vertex to which zero (both "0" and "0*" are included) is assigned is examined to determine whether the input vector assigned to the vertex belongs to the set of minimum vectors or not in the following way.

As in subroutine CMNL, a vertex B under examination is pointed by PTRB. The values assigned to all the vertices which are connected to

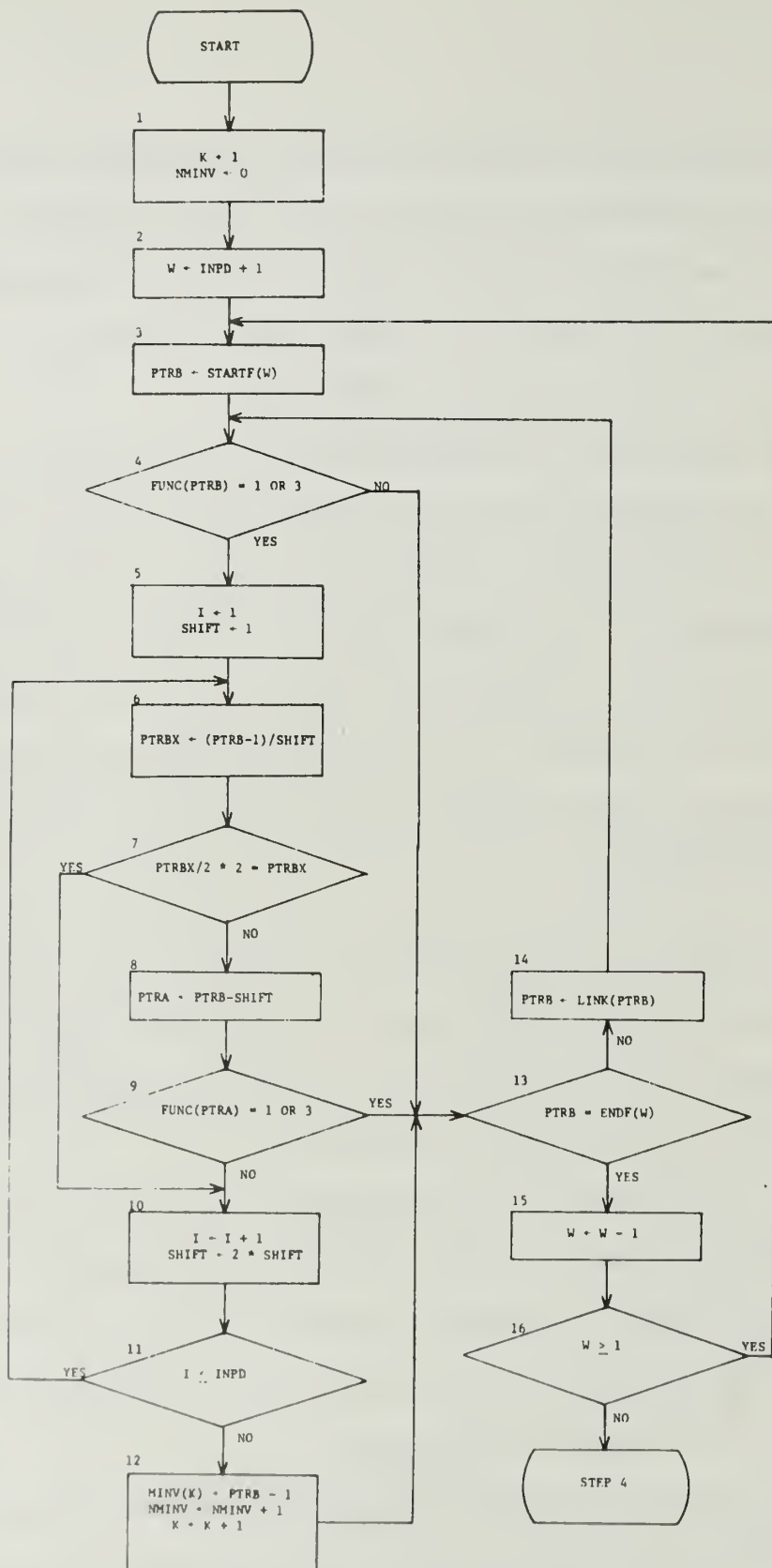


Fig. 3.3.20 Flowchart of subroutine IMC. (Step 3)

vertex B by the edges from vertex B to these vertices are examined in the loop which consists of blocks 6,...,11 and if all these vertices take value "1," the input vector assigned to vertex B is added to the set of minimum vectors.

After the implementation of step 3, the minimum vectors are obtained in array MINV and the number of the minimum vectors are obtained in variable NMINV.

Step 4: This step obtains the subset of the set of minimum vectors which covers all the vertices with "0*" (original zero) and this subset is called a semi-irredundant subset. Flowchart is shown in Fig. 3.3.21.

I is an index which points to a minimum vector in array MINV.

J is an index which points to a vertex in the Large-cube.

In block 2, the first minimum vector MINV(1) is picked up and put into the semi-irredundant subset. In the loop which consists of blocks 4, 5,...,11, the FUNC fields of all the vertices in the Large-cube are examined in the order stored in the array FUNC. The input vector of each vertex with "0*" (original zero) is compared with MINV(1) in block 6 and if MINV(1) covers the vertex with "0*," the LINK field of the vertex is increased by one. (From now on, since we do not use the cube structure constructed in step 1, LINK field is used for storing the degree which indicates how many corresponding vertices with "0*" are covered by a subset of the set of minimum vectors. The LINK field is initialized to zero in block 1.)

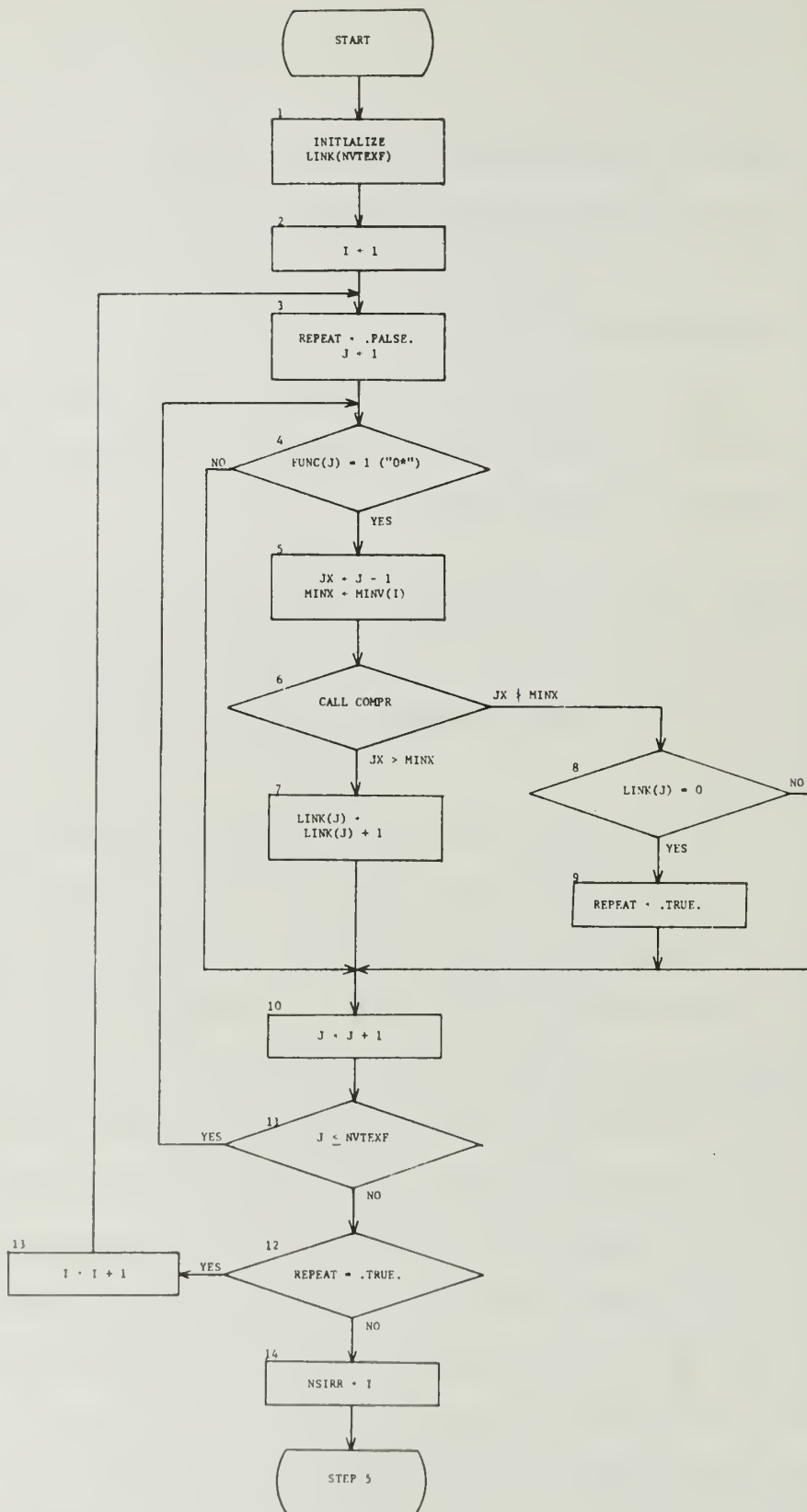


Fig. 3.3.21 Flowchart of subroutine IMC. (Step 4)

After exhausting all the vertices in the Large-cube, if there exists a vertex with "0*" which is not covered at all (LINK field = 0), the next minimum vector is picked up in block 13 and the program control returns to block 3. The same procedure is repeated until all the vertices with "0*" are covered with a subset of the set of minimum vectors.

Since the set of minimum vectors is constructed such that it covers all the zero's in Large-cube (both "0" and "0*" are included), it is obvious that the semi-irredundant subset obtained in the above procedure can cover all the vertices with "0*."

NSIRR stores the number of input vectors in the semi-irredundant subset obtained in the above procedure.

Step 5: This step obtains an irredundant subset from the semi-irredundant subset obtained in step 4. Flowchart is shown in Fig. 3.3.22. In this step, I and J are defined in the same way as in step 4.

In block 1, the first minimum vector in the semi-irredundant subset, MINV(1), is picked up and is tried to be removed from the semi-irredundant subset. This is implemented in the following way. In the loop which consists of blocks 3, 4, ..., 9, the FUNC fields of all the vertices in the Large-cube are examined and the input vector of each vertex with "0*" is compared with MINV(1).

If MINV(1) covers the vertex with "0*," the LINK field of the vertex is decreased by one because we are now trying to remove MINV(1) from the semi-irredundant subset. In block 7, the value of the resulting LINK field is examined and if the value is zero (This means if we

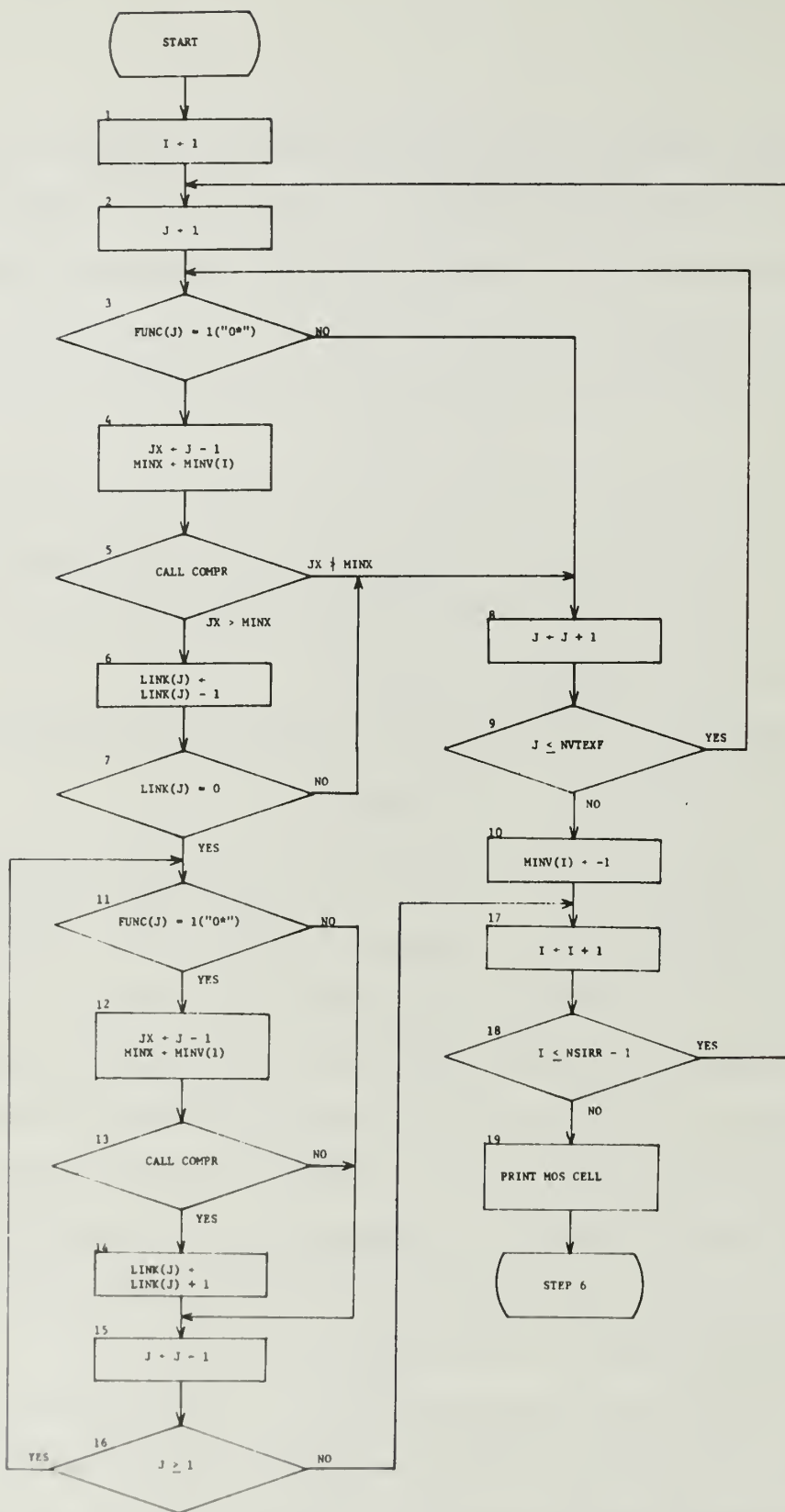


Fig. 3.3.22 Flowchart of subroutine IMC. (Step 5)

remove MINV(1) from the semi-irredundant subset, the resulting subset does not cover all the vertices with "0*" any longer.), MINV(1) cannot be removed from the semi-irredundant subset. The LINK fields which has been decreased in the loop which consists of blocks 3,...,9 have to be restored in the loop which consists of blocks 11,...,16 by increasing them by one. After the restoring operation, the next minimum vector is picked up in block 17.

After exhausting all the vertices in the Large-cube comparing with MINV(1), if all the vertices with "0*" are still covered with the semi-irredundant subset (There exist no vertices with "0*" whose LINK fields take the value zero.), MINV(1) can be removed from the semi-irredundant subset by assigning negative value to MINV(1) (block 10). The next minimum vector is picked up in block 17 and the program control returns to block 2.

The above process is repeated until all the input vectors except the last one in the semi-irredundant subset are exhausted. The resulting subset is called the irredundant subset which realizes an irredundant MOS cell configuration. Finally, in block 19, the irredundant MOS cell configuration is printed (The output format is described in detail in Section 5.1.).

The number of input vectors in the irredundant subset is obtained in a variable NIRR.

Step 6: This step stores the function value realized by the MOS cell obtained in step 5 to the II-th most significant bit of the LABEL field in the N-cube. Two flowcharts are shown in Figs. 3.3.23 and 3.3.24.

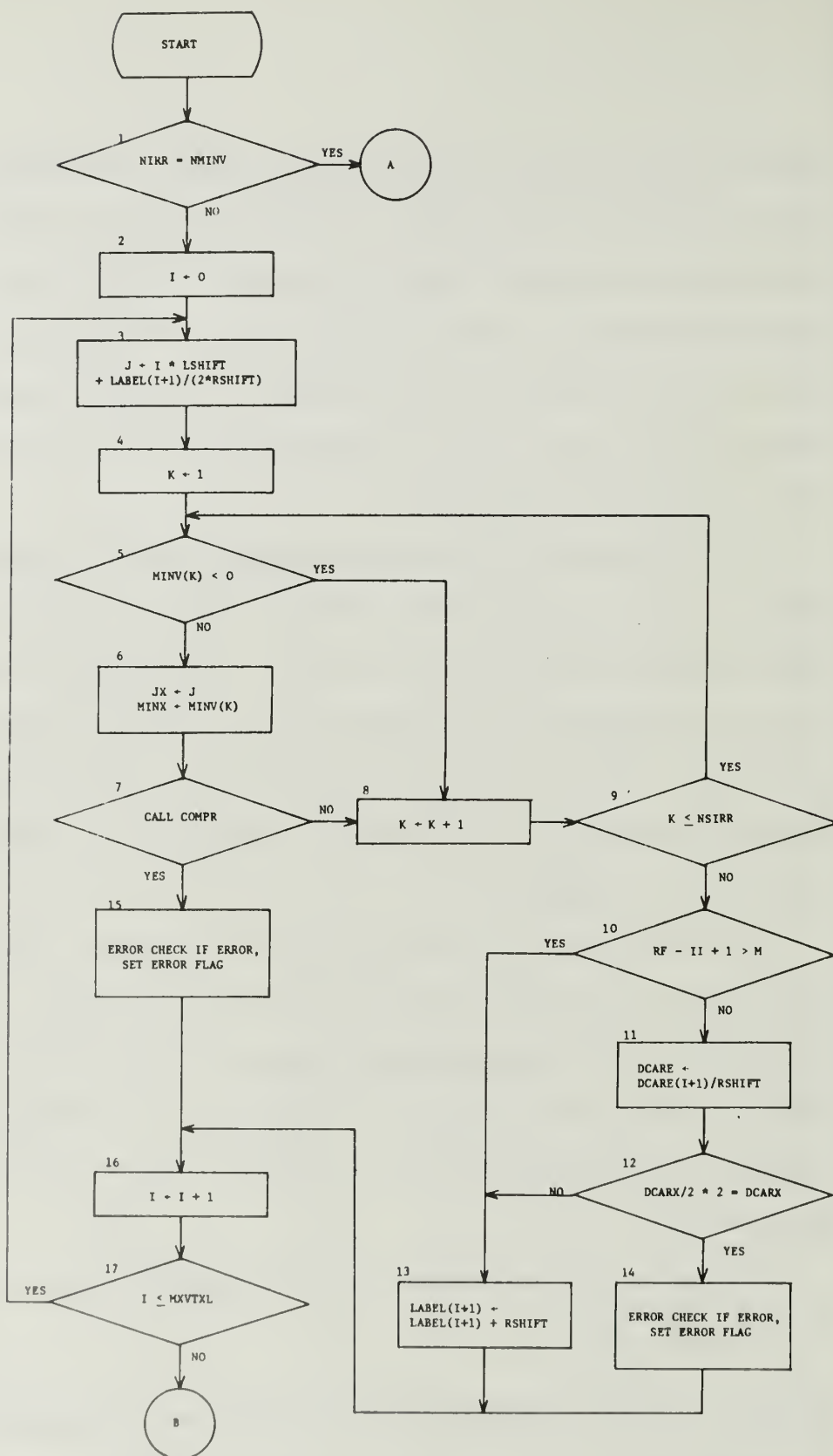


Fig. 3.3.23 Flowchart of subroutine IMC. (Step 6A)

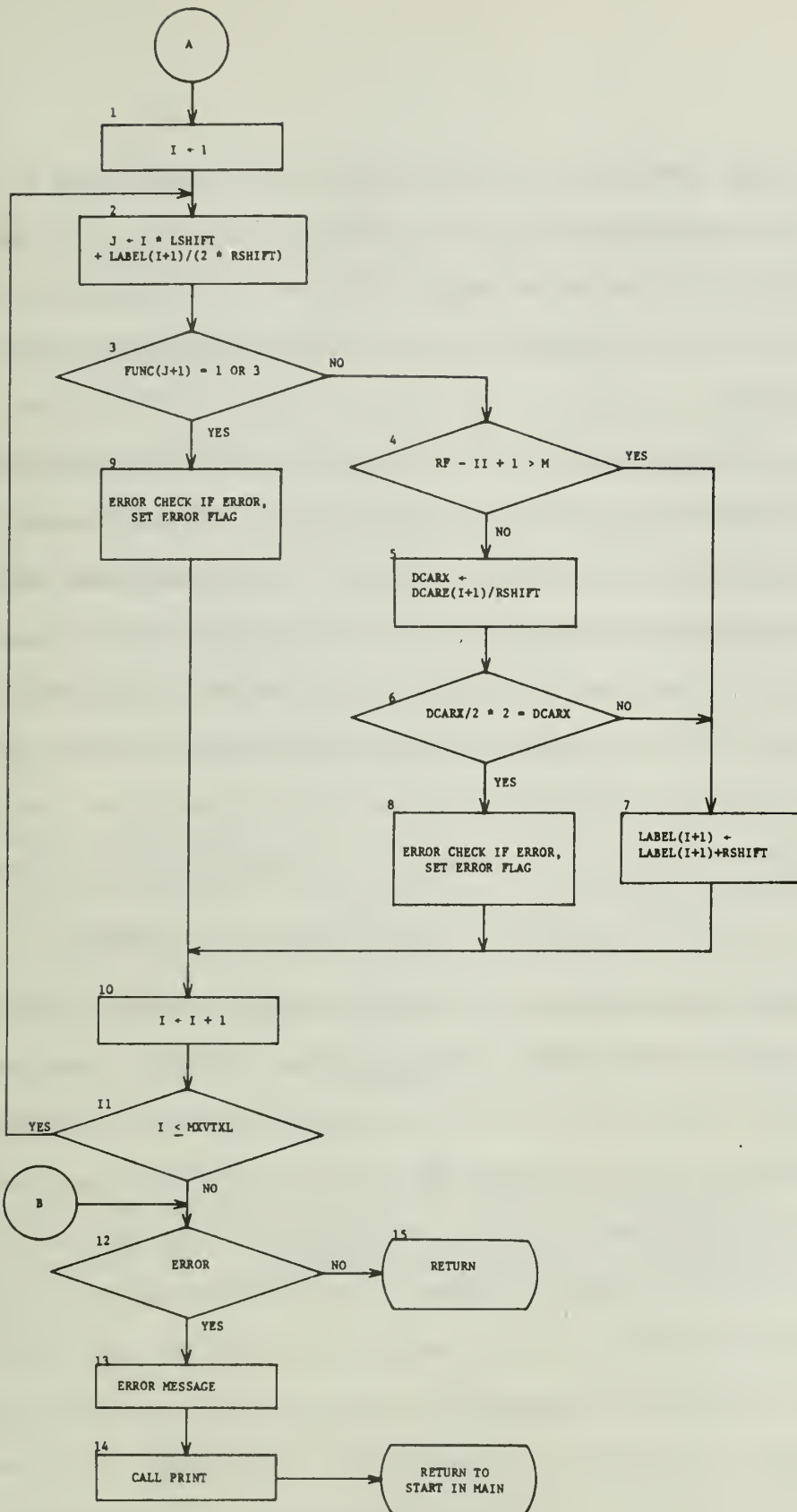


Fig. 3.3.24 Flowchart of subroutine IMC. (Step 6B)

In these flowcharts, I is the index of the program loop and at the same time I represents the vertex number in the N-cube. J is the variable which stores the vertex number of the vertex in the Large-cube to which the maximum permissible function (which is completely specified) is assigned.

In block 1 (Fig. 3.3.23), the dimension of the irredundant subset is compared with the dimension of the set of minimum vectors and if they coincide, the function realized by the irredundant MOS cell takes the same values as those assigned to the Large-cube in step 2. Therefore, in this case, the function value which has been obtained at vertex J in the Large-cube is just copied into the II-th most significant bit of the LABEL field of vertex I in the N-cube. As described in subroutine MPF, there exists the following relation between I and J;

$$J = I \times \text{LSHIFT} + \text{LABEL}(I+1) / (2 * \text{RSHIFT})$$

The above procedure is described in flowchart in Fig. 3.3.24.

On the other hand, in block 1 (Fig. 3.3.23), if they take different values (this occurs in most cases), values of the function realized by the irredundant MOS cell are different from those assigned to the Large-cube in Step 2. Flowchart for this case is shown in Fig. 3.3.23. In this flowchart, after determining the function value realized at vertex J in the Large-cube by the MOS cell obtained in step 5 (this is accomplished by the loop which consists of blocks 5,6, ...,9), it is stored in the II-th most significant bit of the LABEL field of vertex I in the N-cube (block 13). If the function value is zero no action is taken because zero has already been stored.

When the above function value is stored in a bit of the LABEL field in which the output function value has already been stored, the two function values are compared. If they do not coincide, an error flag is set (blocks 14,15).

After storing all the function values in the II-th most significant bit of LABEL field, the status of an error flag is examined in block 12 (Fig. 3.3.24) and if an error flag is on, an error message and the contents of array LABEL are printed. After the contents of array LABEL is printed by calling subroutine PRINT, the program control returns to the START of subroutine MAIN and the next problem is read in. If there is no error, the control returns to subroutine MAIN.

(12) Other Subroutines

(a) Subroutine COMPR - In this subroutine, two input vectors JX and MINX are compared bit by bit and if every bit in JX is larger than or equal to the corresponding bit in MINX, the program control takes the normal return. Otherwise the control is transferred to the statement number shown in the parameter list.

(b) Subroutines ASFUN1 and ASFUN2 - These subroutines are explained in Section 3.2. Flowcharts are shown in Fig. 3.3.25 and Fig. 3.3.26.

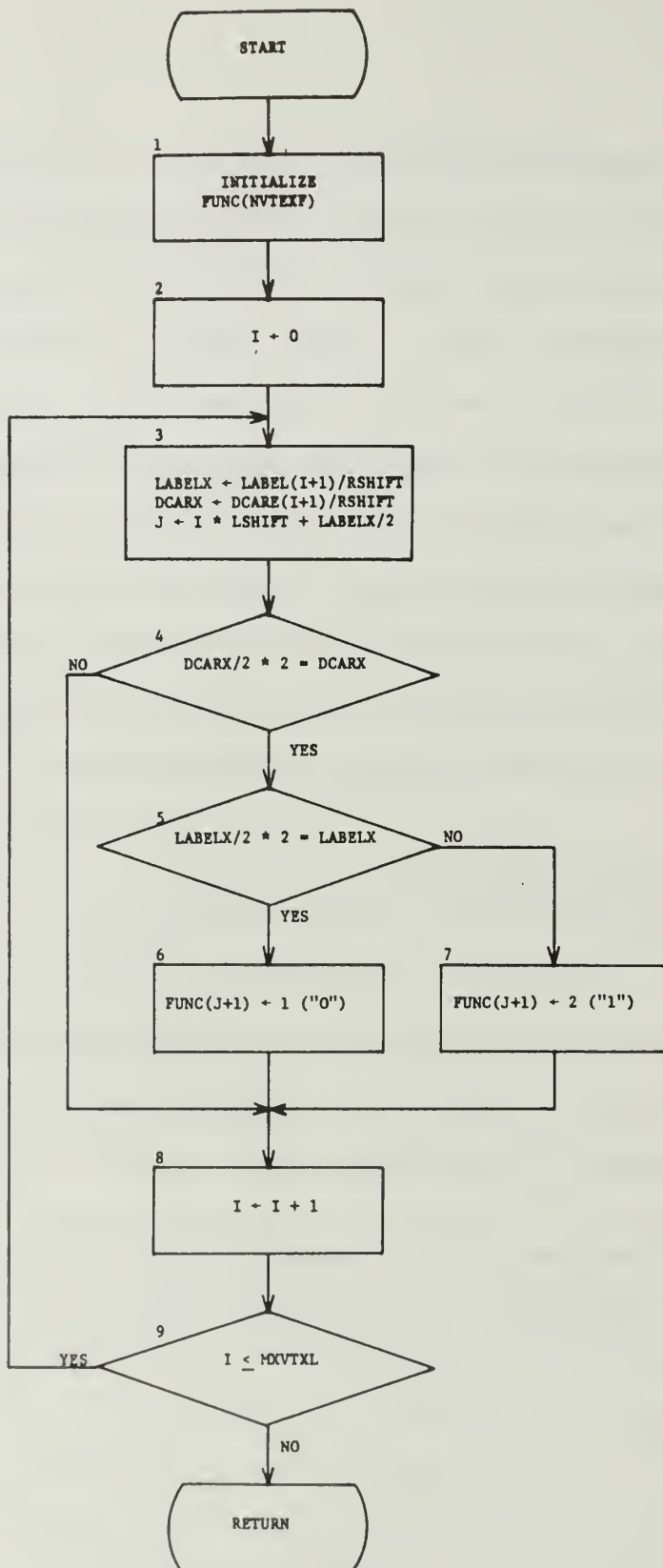


Fig. 3.3.25 Flowchart of subroutine ASFUN1.

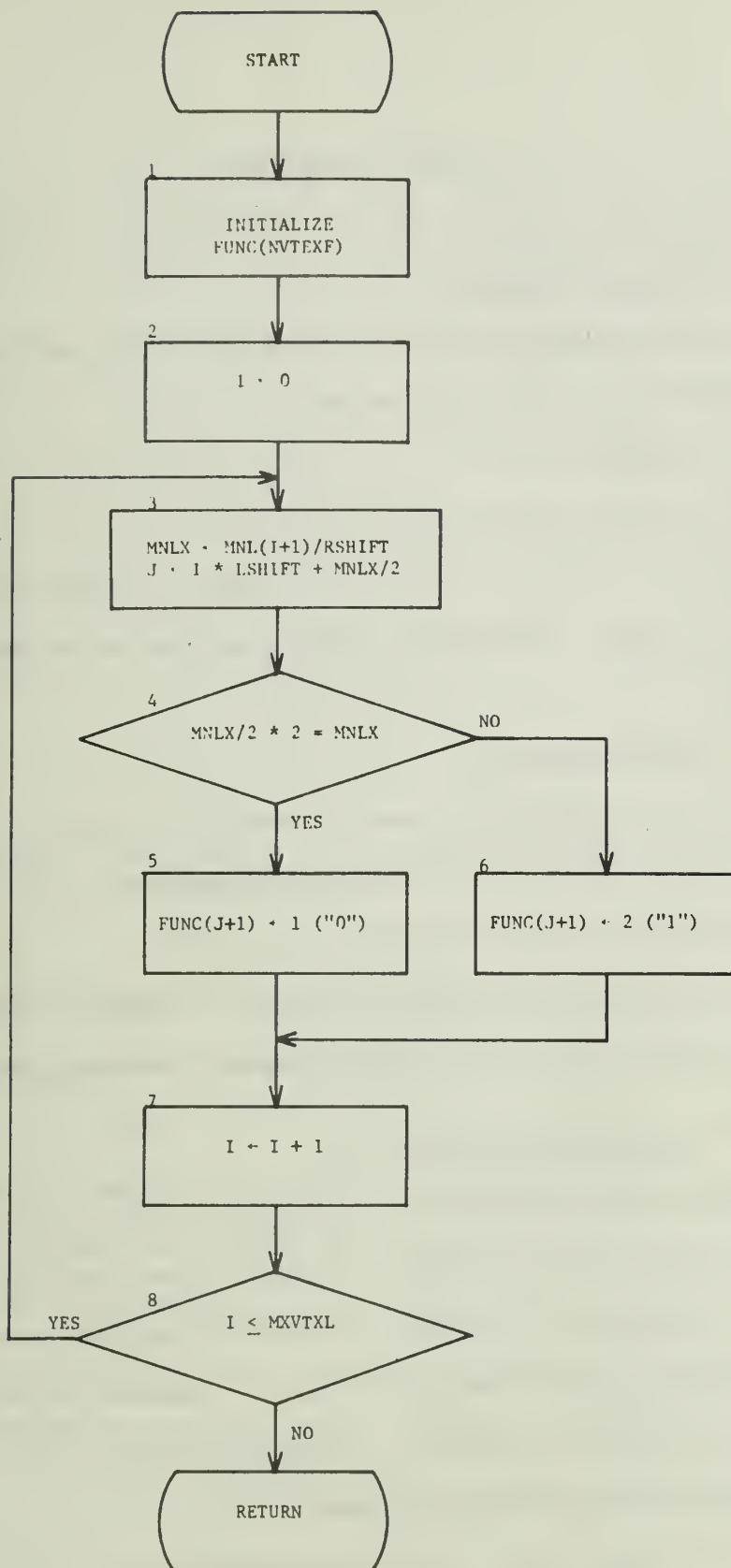


Fig. 3.3.26 Flowchart of subroutine ASFUN2.

4. INPUT DATA SETUP

4.1 Input Data Card Format

For each separate problem, a set of input data cards must be submitted which consist of the following.

(i) < parameter card >

(ii) < output function card > s

(i) will always consist of only a single card, but (ii) may consist of more than one card. The format of each data card is explained in the following.

(i) < parameter card >

This card specifies the number of external variables, N, and the number of output functions, M, for a given problem. The parameter N and M are specified in the following two fields.

Cols. 1-2: This field specifies an integer, N, which is right justified.

Cols. 3-5: This field specifies an integer, M, which is right justified.

(ii) < output function card > s

Although output functions are submitted to this program in the truth table form, the input vector for each output function value is implicitly specified by the order in which the function values are specified in each output function card(s). Depending on the function value (one, zero or don't care), a different character ("1," "0" or "*") is punched on the corresponding column.

The truth table with N external variables and M output functions is shown in Fig. 4.1. This truth table is submitted to the program DIMN in the following way.

x_1	x_2	\dots	x_N	f_1	f_2	--	--	f_M
0	0	\dots	0	f_1^0	f_2^0	--	--	f_M^0
0	0	\dots	1	f_1^1	f_2^1	--	--	f_M^1
		\ddots		\vdots	\vdots			\vdots
				\vdots	\vdots			\vdots
				\vdots	\vdots			\vdots
				\vdots	\vdots			\vdots
				\vdots	\vdots			\vdots
				\vdots	\vdots			\vdots
1	1	\dots	0	$f_1^{2^N-2}$	$f_2^{2^N-2}$	--	--	$f_M^{2^N-2}$
1	1	\dots	1	$f_1^{2^N-1}$	$f_2^{2^N-1}$	--	--	$f_M^{2^N-1}$

Fig. 4.1 Truth table with N external variables and M output functions.

Each output function is specified on one or several output function cards depending upon the number of external variables, N . Since each function value is specified in one column in the output function card(s), if the number of external variables, N , is larger than or equal to 7, the number of input vectors, 2^N , will exceed the number of columns in one card and consequently two or more output function cards are required in order to specify one output function. In fact, the number of cards needed to specify one output function is equal to $\lceil \frac{2^N}{80} \rceil$. (That is, if $N=9$, then $2^9 = 512$. This means that we need 7 cards to specify one output function.)

In Fig. 4.1, f_1^0 , the first function value of function f_1 is specified in the first column of the output function card(s) which specify function f_1 . f_1^1 , the second function value of function f_1 is specified in the second column and so forth. After specifying all the function values of function f_1 , function f_2 is specified on separate output function card(s). The above process is repeated until M output functions are specified on M separate sets of output function card(s). Since blank column terminates one output function specification, (the program DIMN interprets the blank character as the termination sign of one output function specification) the blank character should not be inserted among function specification except at the end of each function specification.

In Fig. 4.2, input data cards for one problem is shown and in

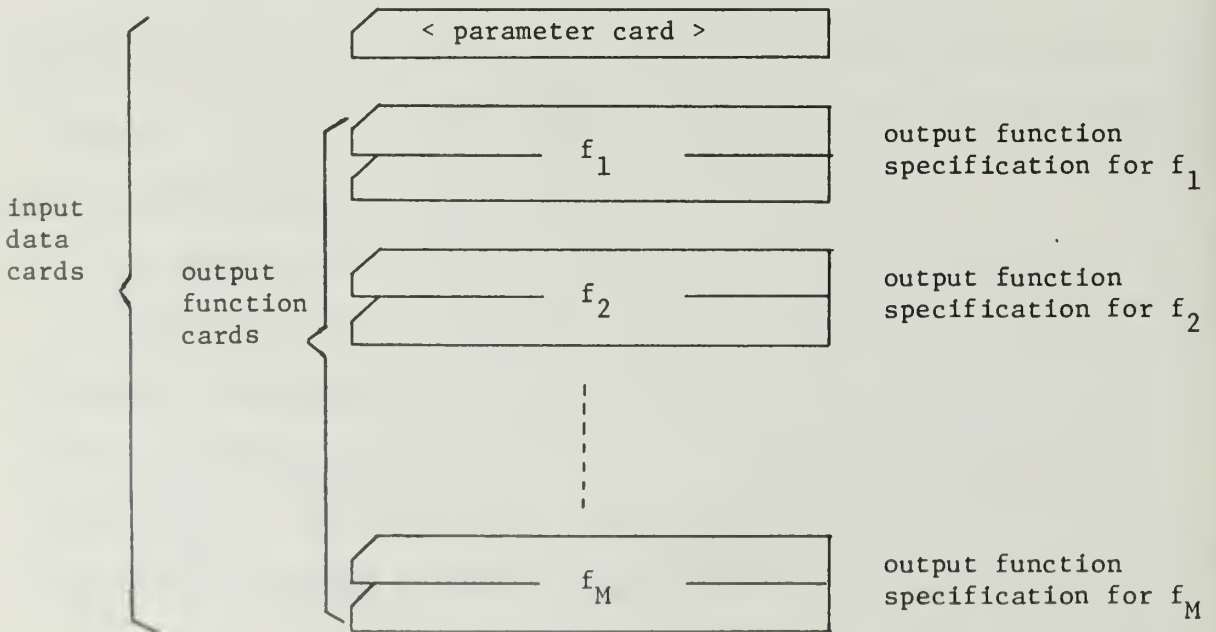


Fig. 4.2 The input data cards for one problem.

Fig. 4.3, an input card sequence for the execution of a typical DIMN problem is shown. We can put as many problems as we want in the fields of these input data cards. If there exists a format error in input data, the following error message is printed out and the program execution halts.

"INPUT ERROR IN DATA CARD I = *** J = *** K = ***"

This message means that an error occurred at the K-th column of the J-th output function card which specifies the I-th output function.

4.2 Restriction on Problem Size

In order to pack problems into a finite amount of space, some restrictions on the size of an acceptable problem are required. In any case, the sum of the number of external variables, N and the number of output functions, M must not exceed 11 and the value of M must not exceed 4. The relation between the maximum number of external variables and the maximum number of output functions which the program DIMN can handle is shown below.

<u>Max. no. of external variables</u>	<u>Max. no. of outputs</u>
10	1
9	2
8	3
7	4

If the above restriction is violated, the following error message is printed and the program execution halts.

"INPUT ERROR IN PARAMETER CARD N = *** M = ***"

```

/* ID   < ID card information >
/* ID  REGION = 200K, TIME = (00.30), LINE = 03000
// EXEC FORTLDGO, REGION. GO = 200K

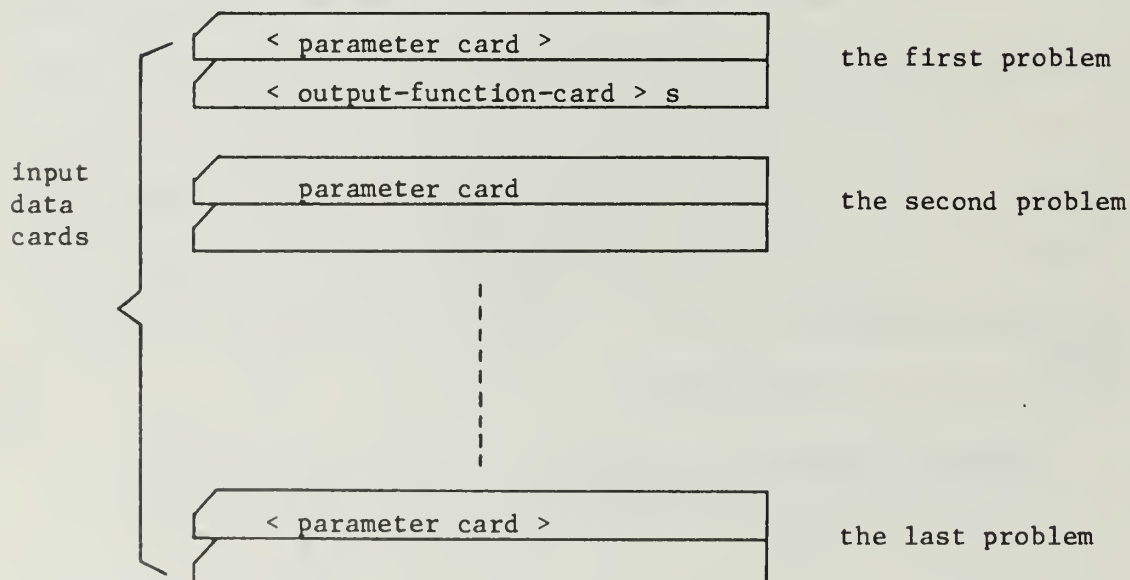
```

FORTTRAN source program

```

/*
// GO. SYSIN DD *

```



```

/*

```

Fig. 4.3 Input card sequence for the execution of a typical DIMN problem.

In this error message, N is the number of external variables and M is the number of output functions on a parameter card. These limitations are essentially imposed by the array sizes in the programs presently written. To loosen the restrictions is mainly a task of increasing array dimensions appropriately.

4.3 Example of Input Data Setup

The following examples show how the input data is set up for the typical DIMN problems.

Example 1: The input data setup for the network with three external variables ($N=3$) and one completely specified output function ($M=1$) is shown in Fig. 4.5. The truth table for the function is shown in Fig. 4.4.

Example 2: The input data setup for the network with three external variables ($N=3$) and two incompletely specified output functions ($M=2$) is shown in Fig. 4.7. The truth table for the functions is shown in Fig. 4.6.

x_1	x_2	x_3	f_1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Fig. 4.4 Truth table for Example 1.

column no.

0 0 0 0 0 0 0 0 0 1 1 1 - - -
1 2 3 4 5 6 7 8 9 0 1 2 - - -

output function
card (f_1)

0 0 1 0 0 0 1 1

parameter card

- 3 - - 1

Fig. 4.5 Possible setup of data cards to specify the problem given in Example 1.

x_1	x_2	x_3	f_1	f_2
0	0	0	1	*
0	0	1	*	0
0	1	0	0	1
0	1	1	1	*
1	0	0	*	0
1	0	1	*	*
1	1	0	0	1
1	1	1	1	1

Fig. 4.6 Truth table for Example 2.

column no. 0 0 0 0 0 0 0 0 0 1 1 1 - - -
 1 2 3 4 5 6 7 8 9 0 1 2 - - -

output function
card for f_2

* 0 1 * 0 * 1 1

output function
card for f_1

1 * 0 1 * * 0 1

parameter card

_ 3 _ _ 2

Fig. 4.7 Possible setup of data cards to specify the problem in Example 2.

5. OUTPUT OF PROCEDURE DIMN

This chapter describes the output of program DIMN. In Section 5.1, the output format is shown for a typical DIMN problem and in Section 5.2, MOS networks obtained by program DIMN are compared with the corresponding MOS networks obtained by previously developed algorithm for several 4 variable functions.

5.1 Output Format

Fig. 5.1 shows the printout obtained by program DIMN for a typical example. In this printout, the subscripted variables X1, X2 - - - - represent the external input variables connected to driver MOS FETs and the subscripted variables U1, U2, - - - - represent the outputs of MOS cells connected to the inputs of other driver MOS FETs.

This printout shows the number of external variables and the number of output functions; two and three, respectively. Then the two output functions are printed in the truth table form where input vectors are implicitly specified by the order in which the function values appear.

In the printout of MOS cell configurations, only driver MOS cells are printed. Fig. 5.2 shows the network obtained from the MOS cell configuration printed in Fig. 5.1. In this network, factoring of literals in switching expressions is done by hand.

Following the above MOS cell configurations, some statistics which are derived from the obtained network are printed. In this example,

```

*****
*
*   DESIGN OF IRREDUNDANT MOS NETWORK   *
*
*****

X = EXTERNAL VARIABLE
U = OUTPUT OF MOS CELL

*****

NUMBER OF EXTERNAL VARIABLES = 3
NUMBER OF OUTPUT FUNCTIONS   = 2

FUNCTION 1
10010101

FUNCTION 2
00100111

NETWORK CONFIGURATION

MOS CELL 1  0-----X2--X3-----0

MOS CELL 2  0--|---X2--U1---|---0
               |---X1--X3---|

MOS CELL 3  0--|---X3--U1--U2---|---0
               |---X2--U1-----|
               |---X1--U2-----|

MOS CELL 4  0-----U2-----0

NUMBER OF MOS CELLS = 4
NUMBER OF MOS FETS  = 14
( WITHOUT FACTORING )

ELAPSED TIME = 0.12 SEC

```

Fig. 5.1 Printout obtained by program DIMN.

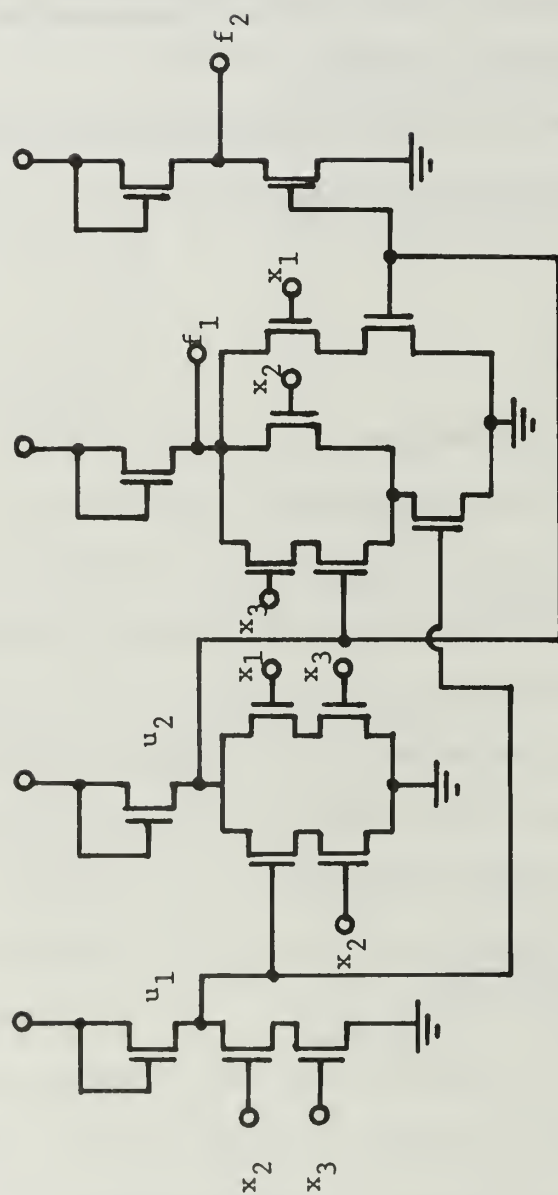


Fig. 5.2 Network obtained from the MOS cell configuration in Fig. 5.1.

the number of MOS cells is four, the number of driver MOS FETs is 14 (without factoring) and elapsed time for constructing the network is 0.14 sec. As a timing subroutine, STEPZ in FORTRAN system subroutines is used.

5.2 Networks Obtained by Program DIMN

The networks obtained by program DIMN are shown in Appendix A for several three and four variable functions. Each page in Appendix A contains the output of program DIMN for one problem in upper half and the resulting network obtained from the above computer output in lower half. In the resulting networks, literals in switching expressions are factored by hand.

Comparing the four variable networks obtained in Appendix A with the corresponding four variable networks obtained in [6] based on previously developed Liu's algorithm, we observe that significant improvement with respect to the number of driver MOS FETs can be achieved by Lai's algorithm on which program DIMN is based.

The number of MOS FETs obtained by two different algorithms for twelve 4-variable single-output functions are compared in Table 5.1.

In this table, functions are represented in hexadecimal form. For example, function $f = 000000000000\ 1011$ is represented in hexadecimal form $f = 000B$. This table shows that the average number of MOS FETs is improved by a factor of 1.36 (from 12.9 to 9.5) by Algorithm DIMN. In the following we can show one typical example for the function $f_1 = 000B$ (hexadecimal representation) with 4 external variables.

Function in hexadecimal	Number of driver MOS FETs in Network	
	obtained by Liu's Algo.	obtained by Lai's Algo. ⁺
0001	5	5
0002	6	5
0006	9	7
0007	5	5
0008	7	5
0009	13	8
000B	12	6
0016	12	10
0069	21	12
0117	9	9
6996	28	21
9669	28	21
Total FETs	155	114
Average FETs in one Network	12.9	9.5

Table 5.1 Number of driver MOS FETs in networks obtained by two different algorithms.

⁺ Program DIMN is based on Lai's algorithm.

The network obtained by program DIMN and the network obtained based on Liu's algorithm are shown in Fig. 5.3 and Fig. 5.4, respectively. As can be seen in these figures, the number of driver MOS FETs for this particular example can be reduced to one half by applying Algorithm DIMN.

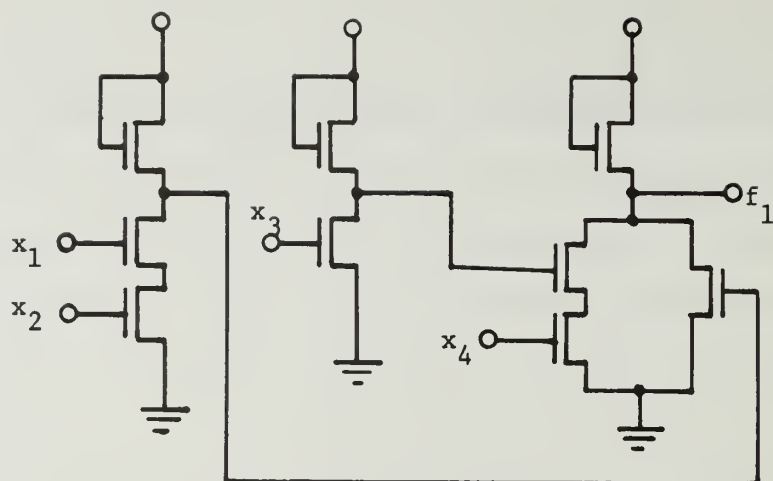


Fig. 5.3 Network obtained by program DIMN for $f_1 = 000B$; 3 load MOS FETs and 6 drivers.

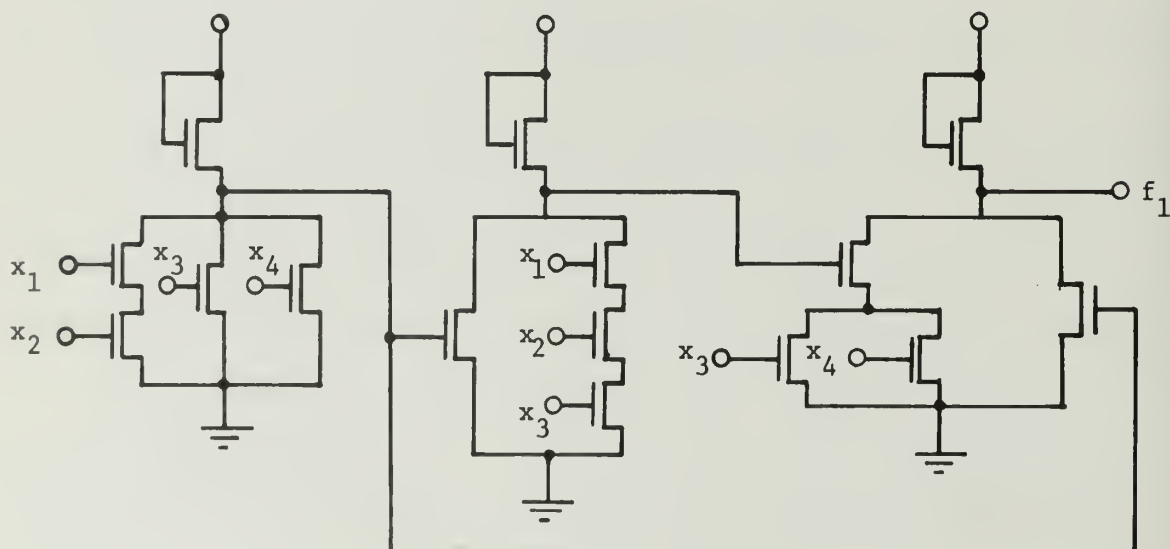


Fig. 5.4 Network obtained applying previously developed algorithm for $f_1 = 000B$; 3 load MOS FETs and 12 drivers.

6. CONCLUSION

As discussed in the introduction, Lai's algorithm extended the previously developed algorithms and made it possible to obtain irredundant MOS networks with minimum number of negative gates (MOS cells). Although Algorithm DIMN does not guarantee exhaustion of all possible irredundant MOS networks for an arbitrary given function, Algorithm DIMN is so far the only existing procedure which can obtain irredundant MOS networks efficiently. Our computational experience with Algorithm DIMN implemented as program DIMN shows the efficiency of this algorithm.

REFERENCES

- [1] Ibaraki, T. and S. Muroga, "Synthesis of Networks With a Minimum Number of Negative Gates," IEEE Transaction on Computers, Vol. C-20, No. 1, January 1971, pp. 49-58.
- [2] Ibaraki, T., "Gate Interconnection Minimization of Switching Networks Using Negative Gates," IEEE Transaction on Computers, Vol. C-20, June 1971, pp. 698-706.
- [3] Liu, T. K., "Synthesis of Logic Networks With MOS Complex Cells," Doctoral thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois.
- [4] Nakamura, K., N. Tokura and T. Kasami, "Minimal Decomposition of a Logical Function into Monotone Functions," The Institute of Electronics and Communication Engineers of Japan, Vol. 54-C, No. 1, January 1971, pp. 18-25.
- [5] Lai, H. C., "Design of Diagnosable Networks," Doctoral thesis (in preparation), Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois.
- [6] Shinozaki, T., "Computer Program for Designing Optimal Networks With MOS Gates," Report No. UIUCDCS-R-72-502, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois.

APPENDIX A

Networks Obtained by Program DIMN

```

*****
*      DESIGN OF IRREDUNDANT MOS NETWORK      *
*      *****

```

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

```

*****

```

NUMBER OF EXTERNAL VARIABLES = 3
 NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
 00100011

NETWORK CONFIGURATION

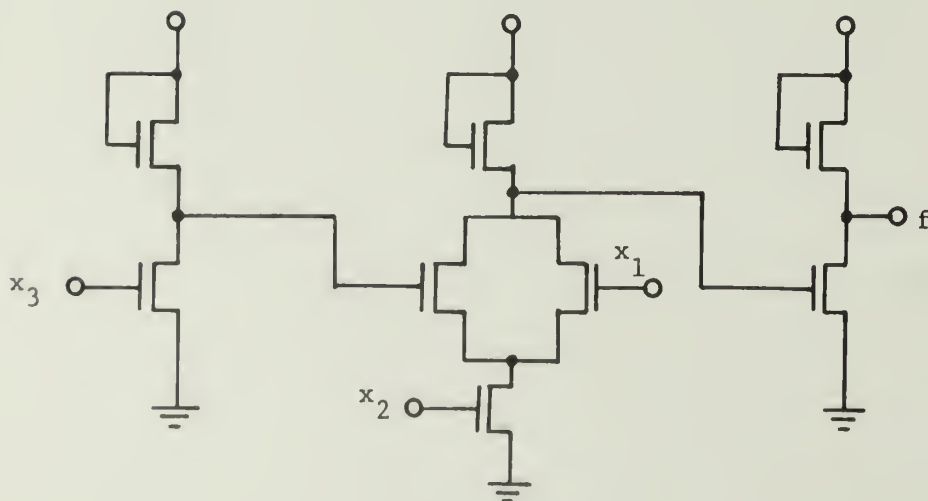
MOS CELL 1 0-----X3-----0

MOS CELL 2 0--|--X2--U1--|--0
 | |--X1--X2--|

MOS CELL 3 0-----U2-----0

NUMBER OF MOS CELLS = 3
 NUMBER OF MOS FETS = 6
 (WITHOUT FACTORING)

ELAPSED TIME = 0.07 SEC



```

*****
*                                     *
*   DESIGN OF IRREDUNDANT MOS NETWORK   *
*                                     *
*****

```

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

```
*****
```

NUMBER OF EXTERNAL VARIABLES = 3
 NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
 0*10***1

NETWORK CONFIGURATION

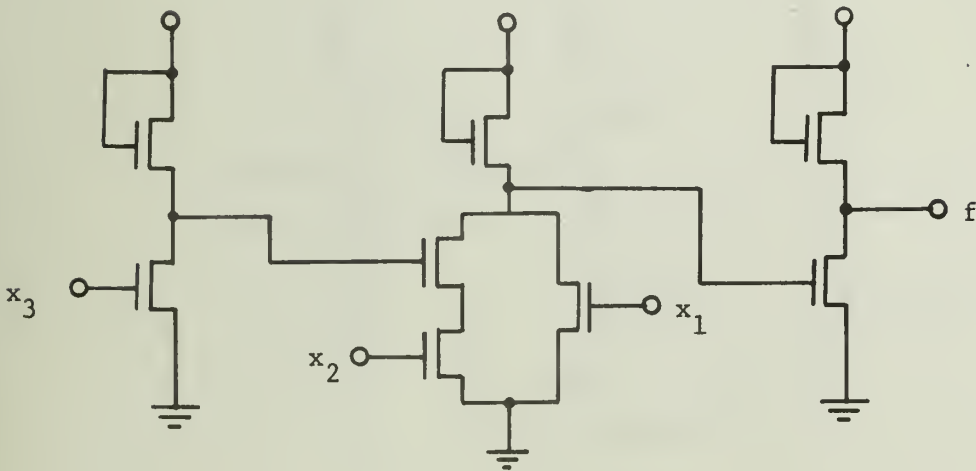
MOS CELL 1 0-----X3-----0

MOS CELL 2 0--|--X2--U1--|--0
 |--X1-----|

MOS CELL 3 0-----U2-----0

NUMBER OF MOS CELLS = 3
 NUMBER OF MOS FETS = 5
 (WITHOUT FACTORING)

ELAPSED TIME = 0.07 SEC



```

*****
*                                     *
*   DESIGN OF IRREDUNDANT MOS NETWORK   *
*                                     *
*****

```

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

```
*****
```

NUMBER OF EXTERNAL VARIABLES = 3
 NUMBER OF OUTPUT FUNCTIONS = 2

FUNCTION 1
 1*01**01

FUNCTION 2
 *01*0*11

NETWORK CONFIGURATION

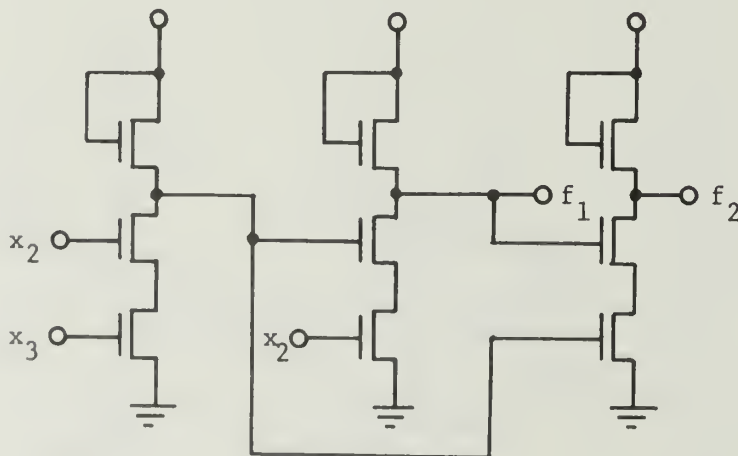
MOS CELL 1 0-----X2--X3-----0

MOS CELL 2 0-----X2--U1-----0

MOS CELL 3 0-----U1--U2-----0

NUMBER OF MOS CELLS = 3
 NUMBER OF MOS FETS = 6
 (WITHOUT FACTORING)

ELAPSED TIME = 0.05 SEC



 * DESIGN OF IRREDUNDANT MOS NETWORK *
 *

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

NUMBER OF EXTERNAL VARIABLES = 4
 NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
 0000000000000001

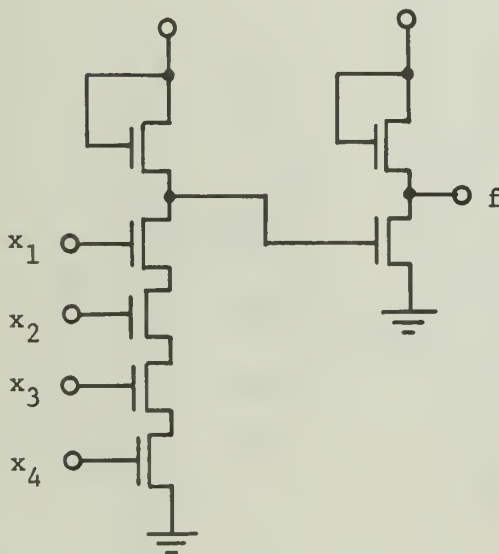
NETWORK CONFIGURATION

MOS CELL 1 0-----X1--X2--X3--X4-----0

MOS CELL 2 0-----U1-----0

NUMBER OF MOS CELLS = 2
 NUMBER OF MOS FETS = 5
 (WITHOUT FACTORING)

ELAPSED TIME = 0.03 SEC



```

*****
*                                     *
*   DESIGN OF IRREDUNDANT MOS NETWORK   *
*                                     *
*****

```

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

```
*****
```

NUMBER OF EXTERNAL VARIABLES = 4
 NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
 0000000000000010

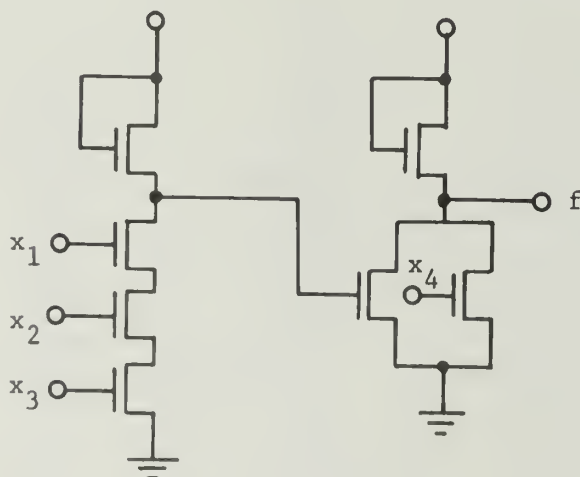
NETWORK CONFIGURATION

MOS CELL 1 0-----X1--X2--X3-----0

MOS CELL 2 0-- $\left| \begin{array}{c} \text{--U1--} \\ \text{--X4--} \end{array} \right|$ --0

NUMBER OF MOS CELLS = 2
 NUMBER OF MOS FETS = 5
 (WITHOUT FACTORING)

ELAPSED TIME = 0.06 SEC



```

*****
*   DESIGN OF IRREDUNDANT MOS NETWORK   *
*   *****                           *

```

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

```

*****

```

NUMBER OF EXTERNAL VARIABLES = 4
 NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
 00000000000000110

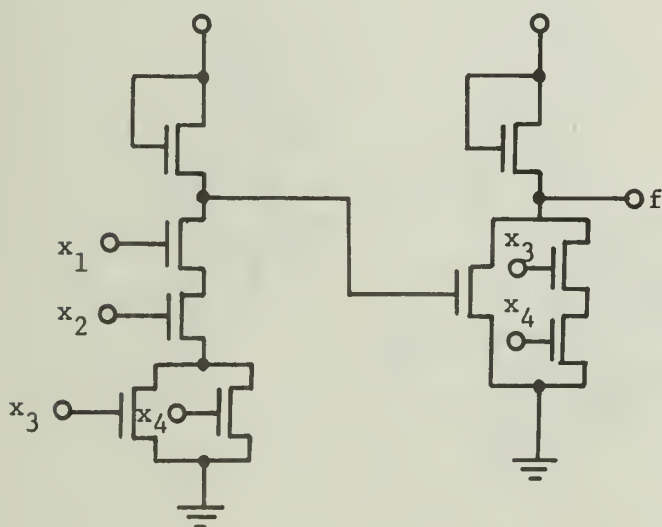
NETWORK CCFNFIGURATION

MOS CELL 1 0--|--X1--X2--X4--|
 |--X1--X2--X3--| --0

MOS CELL 2 0--|--X3--X4--|
 |--U1-----| --0

NUMBER OF MOS CELLS = 2
 NUMBER OF MOS FETS = 9
 (WITHOUT FACTORING)

ELAPSED TIME = 0.06 SEC



```

*****
*
*   DESIGN OF IRREDUNDANT MOS NETWORK
*
*
*****

```

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

NUMBER OF EXTERNAL VARIABLES = 4
 NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
 00000000000000111

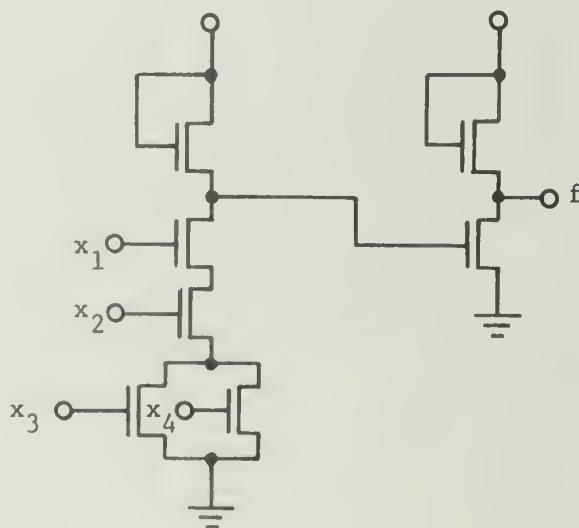
NETWORK CONFIGURATION

MOS CELL 1 0-- $\left| \begin{array}{c} \text{--X1--X2--X4--} \\ \text{--X1--X2--X3--} \end{array} \right|$ --0

MOS CELL 2 0-----U1-----0

NUMBER OF MOS CELLS = 2
 NUMBER OF MOS FETS = 7
 (WITHOUT FACTORING)

ELAPSED TIME = 0.07 SEC



```

*****
*      DESIGN OF IRREDUNDANT MOS NETWORK      *
*      *****

```

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

```
*****
```

NUMBER OF EXTERNAL VARIABLES = 4
 NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
 0000000000001000

NETWORK CONFIGURATION

MOS CELL 1 0-----X1--X2-----0

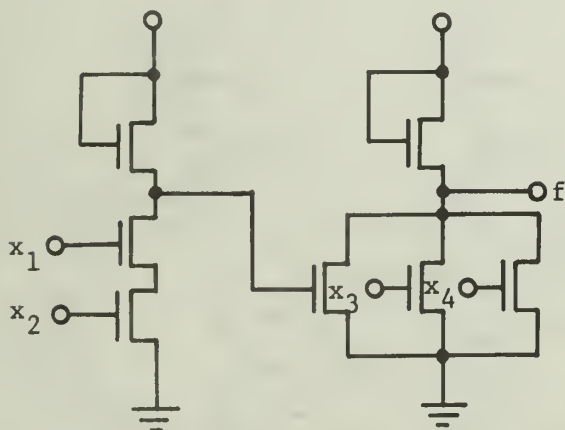
MOS CELL 2 0--

--U1--
--X4--
--X3--

--0

NUMBER OF MOS CELLS = 2
 NUMBER OF MOS FETS = 5
 (WITHOUT FACTORING)

ELAPSED TIME = 0.09 SEC



```

*****
*          DESIGN OF IRREDUNDANT MOS NETWORK          *
*          *****

```

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

```

*****

```

NUMBER OF EXTERNAL VARIABLES = 4
 NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
 0000000000001001

NETWORK CONFIGURATION

MOS CELL 1 0-----X1--X2-----0

MOS CELL 2 0-----X3--X4-----0

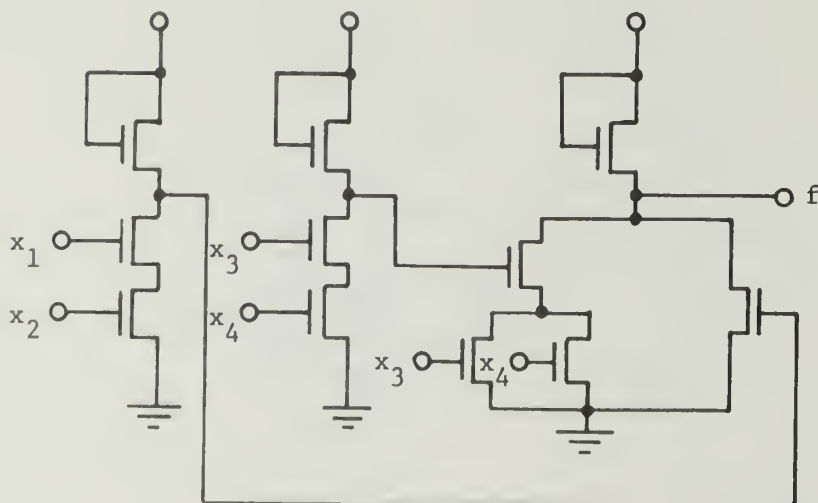
MOS CELL 3

---X4---U2---
---X3---U2---
---U1-----

 ---0

NUMBER OF MOS CELLS = 3
 NUMBER OF MOS FETS = 9
 (WITHOUT FACTORING)

ELAPSED TIME = 0.15 SEC



 * DESIGN OF IRREDUNDANT MOS NETWORK *
 * *****

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

NUMBER OF EXTERNAL VARIABLES = 4
 NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
 0000000000001011

NETWORK CONFIGURATION

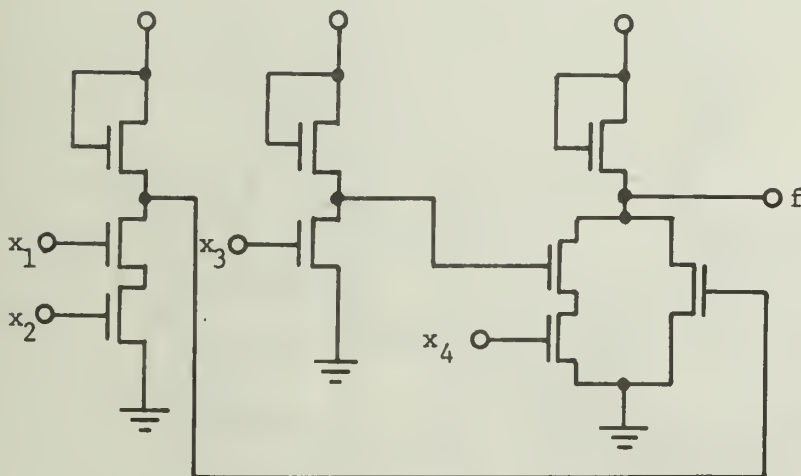
MOS CELL 1 0-----X1--X2-----0

MOS CELL 2 0-----X3-----0

MOS CELL 3 0--|--X4--U2--|--0
 --U1-----|

NUMBER OF MOS CELLS = 3
 NUMBER OF MOS FETS = 6
 (WITHOUT FACTORING)

ELAPSED TIME = 0.15 SEC



 *
 * DESIGN OF IRREDUNDANT MOS NETWORK *
 *

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

NUMBER OF EXTERNAL VARIABLES = 4
 NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
 0000000000010110

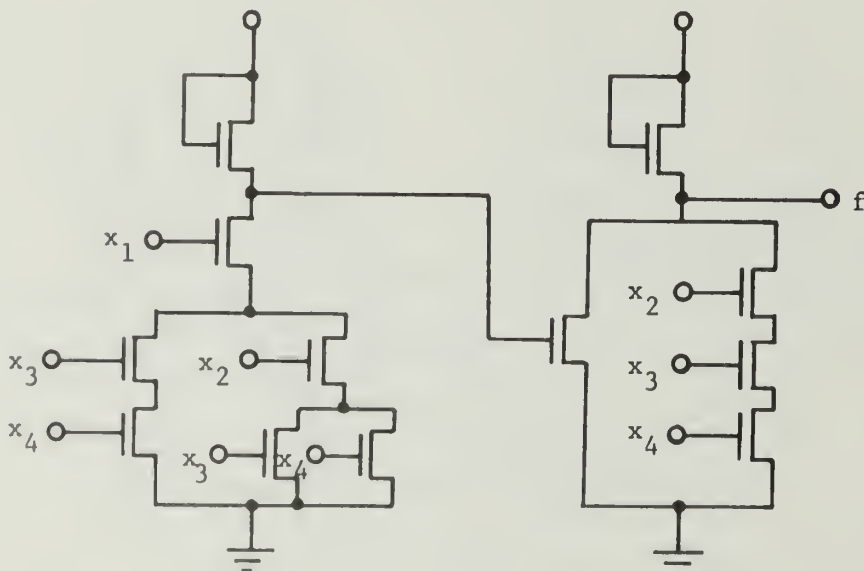
NETWORK CONFIGURATION

MOS CELL 1 0-- $\left| \begin{array}{l} \text{--X1--X3--X4--} \\ \text{--X1--X2--X4--} \\ \text{--X1--X2--X3--} \end{array} \right| \text{--0}$

MOS CELL 2 0-- $\left| \begin{array}{l} \text{--X2--X3--X4--} \\ \text{--U1-----} \end{array} \right| \text{--0}$

NUMBER OF MOS CELLS = 2
 NUMBER OF MOS FETS = 13
 (WITHOUT FACTORING)

ELAPSED TIME = 0.09 SEC



 *
 * DESIGN OF IRREDUNDANT MOS NETWORK *
 *

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

NUMBER OF EXTERNAL VARIABLES = 4
 NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
 0000000001101001

NETWORK CONFIGURATION

MOS CELL 1 0-----X3--X4-----0

MOS CELL 2 0--

--X1--X4--U1--
--X1--X3--U1--
--X1--X2-----

--0

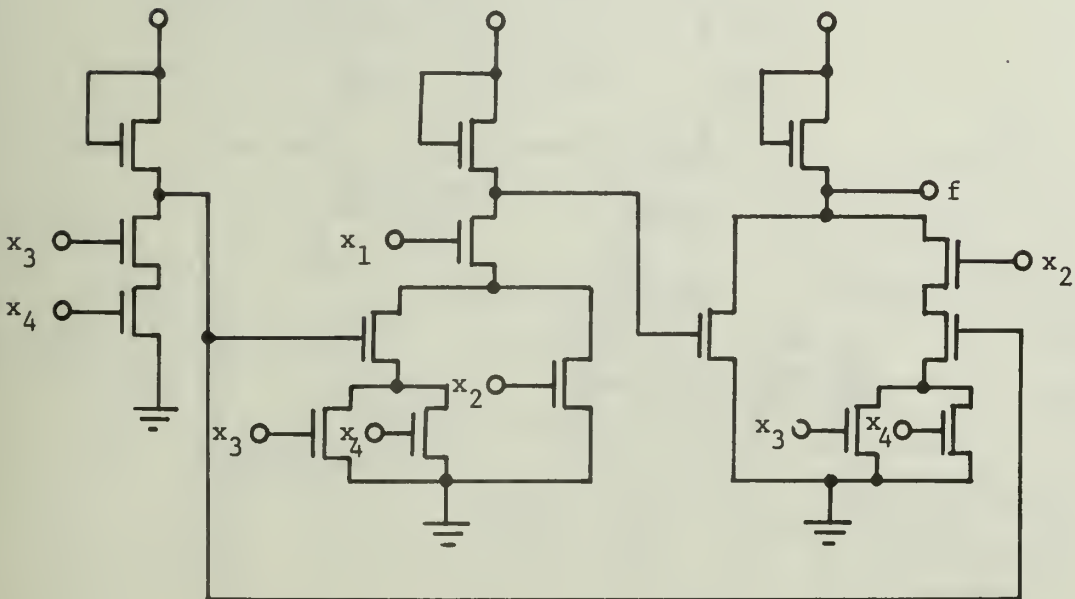
MOS CELL 3 0--

--X2--X4--U1--
--X2--X3--U1--
--U2-----

--0

NUMBER OF MOS CELLS = 3
 NUMBER OF MOS FETS = 17
 (WITHOUT FACTORING)

ELAPSED TIME = 0.19 SEC



 *
 * DESIGN OF IRREDUNDANT MOS NETWORK *
 *

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

NUMBER OF EXTERNAL VARIABLES = 4
 NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
 0000000100010111

NETWORK CONFIGURATION

MOS CELL 1 0--

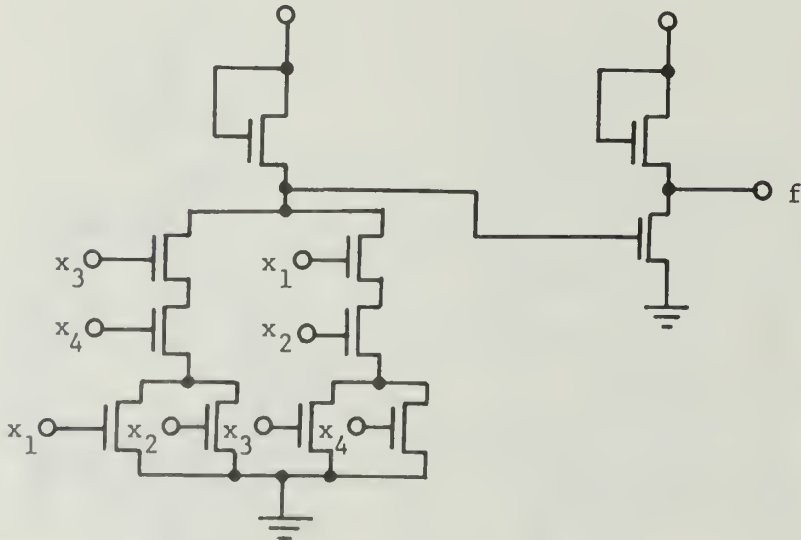
--X2--X3--X4--
--X1--X3--X4--
--X1--X2--X4--
--X1--X2--X3--

--0

MOS CELL 2 0-----U1-----0

NUMBER OF MOS CELLS = 2
 NUMBER OF MOS FETS = 13
 (WITHOUT FACTORING)

ELAPSED TIME = 0.07 SEC



 * DESIGN OF IRREDUNDANT MOS NETWORK *
 *

101

X = EXTERNAL VARIABLE
 U = OUTPUT OF MOS CELL

NUMBER OF EXTERNAL VARIABLES = 4
 NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
 0110100110010110

NETWORK CONFIGURATION

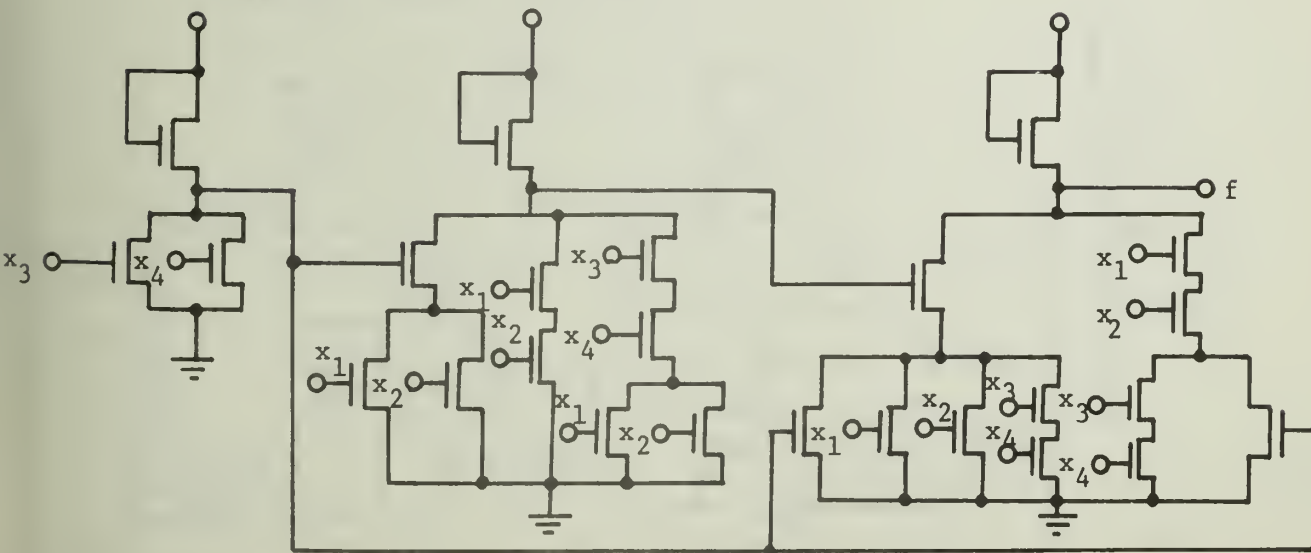
MOS CELL 1 0-- $\left| \begin{array}{c} \text{--X4--} \\ \text{--X3--} \end{array} \right| \text{--0}$

MOS CELL 2 0-- $\left| \begin{array}{c} \text{--X2--X3--X4--} \\ \text{--X1--X3--X4--} \\ \text{--X2--U1-----} \\ \text{--X1--U1-----} \\ \text{--X1--X2-----} \end{array} \right| \text{--0}$

MOS CELL 3 0-- $\left| \begin{array}{c} \text{--X1--X2--X3--X4--} \\ \text{--X3--X4--U2---} \\ \text{--X1--X2--U1---} \\ \text{--U1--U2-----} \\ \text{--X2--U2-----} \\ \text{--X1--U2-----} \end{array} \right| \text{--0}$

NUMBER OF MOS CELLS = 3
 NUMBER OF MOS FETS = 30
 (WITHOUT FACTORING)

ELAPSED TIME = 0.25 SEC



X = EXTERNAL VARIABLE
U = OUTPUT OF MOS CELL

NUMBER OF EXTERNAL VARIABLES = 4
NUMBER OF OUTPUT FUNCTIONS = 1

FUNCTION 1
1001011001101001

NETWORK CONFIGURATION

MOS CELL 1 0-----X3--X4-----0

```

MOS CELL 2  0--|--X2--X4--U1--|
                |--X2--X3--U1--|
                |--X1--X4--U1--|--0
                |--X1--X3--U1--|
                |--X1--X2-----|

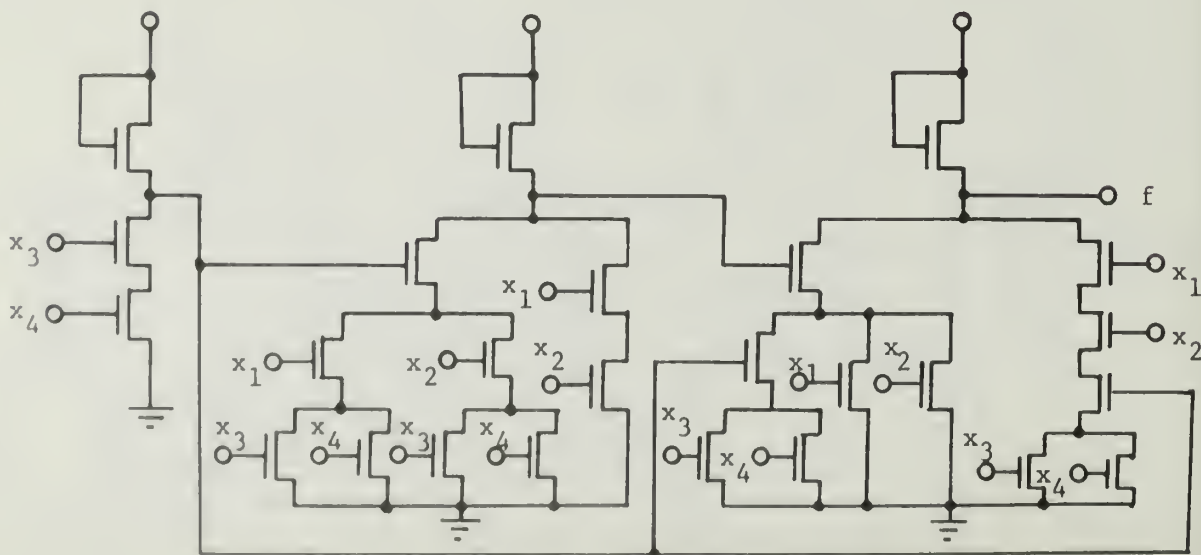
```

MOS CELL 3 0--

--X1--X2--X4--U1--	
--X1--X2--X3--U1--	
--X4--U1--U2----	
--X3--U1--U2----	--0
--X2--U2-----	
--X1--U2-----	

NUMBER OF MOS CELLS = 3
NUMBER OF MOS FETS = 34
(WITHOUT FACTORING)

ELAPSED TIME = 0.24 SEC



APPENDIX B

Program Listing

THIS PROGRAM IS THE IMPLEMENTATION OF ALGORITHM DIMN (DESIGN OF IRREDUNDANT MOS NETWORK).
THIS PROGRAM IS APPLICABLE TO THE NETWORKS WITH MULTIPLE INCOMPLETELY SPECIFIED OUTPUTS.
THE RELATION BETWEEN THE MAXIMUM NUMBER OF EXTERNAL VARIABLES AND THE MAXIMUM NUMBER OF OUTPUTS WHICH THIS PROGRAM CAN HANDLE IS SHOWN BELOW.

MAX. EXTERNAL VARIABLES

10
9
8
7

MAX. OUTPUTS

1
2
3
4

THIS PROGRAM IS CONSTRUCTED WITH THE FOLLOWING MAIN PUBROUTINES AND THE OTHER SMALL SUBROUTINES.

SUBROUTINE NAME OPERATION

NCUBE CONSTRUCT N-CUBE ACCORDING TO THE NUMBER OF EXTERNAL VARIABLES.

INPUT READ DATA INTO N-CUBE (LABEL, DCARE).

CMNL IMPLEMENT CONDITIONAL MINIMUM LABELING.

CMXL IMPLEMENT CONDITIONAL MAXIMUM LABELING.

MPF OBTAIN MAXIMUM PERMISSIBLE FUNCTION FROM

MINIMUM LABEL AND MAXIMUM LABEL.

IMC OBTAIN IRREDUNDANT MOS CELL CONFIGURATION

FROM MAXIMUM PERMISSIBLE FUNCTION.

IMPLICIT INTEGER(A-Z)
INTEGER*2 LABEL ,DCARE ,MNL ,MXL ,CHAIN ,FUNC ,LINK

1 COMMON ,MINV

1 COMMON II ,N ,M ,RF

1 ,LABEL(1024) ,DCARE(1024) ,MNL(1024) ,MXL(1024)

2 ,CHAIN(1024) ,STARTL(11) ,ENDL(11) ,FUNC(16384)

3 ,LINK(16384) ,MINV(16384) ,STARTF(15) ,ENDF(15)

LOGICAL CS

REAL UTIME

II POINTER WHICH INDICATES THE CURRENT STEP.

N NUMBER OF EXTERNAL VARIABLES.

M NUMBER OF OUTPUT FUNCTIONS.

RF MINIMUM NUMBER OF MOS CELLS

CS IS SET IN SUBROUTINE-INPUT IF THE GIVEN OUTPUT

FUNCTIONS ARE COMPLETELY SPECIFIED.

UTIME STORES THE TIME ELAPSED FOR CONSTRUCTING NETWORK.

TFET STORES TOTAL NUMBER OF DRIVER MOS FETS IN NETWORK.

LABEL(1024) STORES LABEL AND FUNCTION VALUE WHICH IS ASSIGNED

TO EACH VERTEX IN N-CUBE.

DCARE(1024) STORES THE DONTCARE BIT OF OUTPUT FUNCTIONS.

MNL (1024) STORES THE LABEL ASSIGNED TO EACH VERTEX BY

CONDITIONAL MINIMUM LABELING.

MXL (1024) STORES THE LABEL ASSIGNED TO EACH VERTEX BY

CONDITIONAL MAXIMUM LABELING.

CHAIN(1024) STORES THE LINK TO THE NEXT VERTEX WITH THE SAME

WEIGHT IN N-CUBE.

STARTL(11) POINTER TO THE FIRST VERTEX WITH THE SAME WEIGHT

IN N-CUBE.

ENDL (11) POINTER TO THE LAST VERTEX WITH THE SAME WEIGHT IN

N-CUBE.

FUNC(16384) STORES MAXIMUM PERMISSIBLE FUNCTION ASSIGNED TO EACH

VERTEX IN LARGE CUBE FOR OBTAINING IRREDUNDANT

```

C      MOS CELL CONFIGURATION.
C      LINK(16384)      STORES THE LINK TO THE NEXT VERTEX WITH THE SAME
C      MINV(16384)      WEIRHT IN LARGE CUBE.
C                        STORES THE MINIMUM VECTOR OBTAINED IN THE PROCESS
C                        OF IMC.
C      CONTINUE
C      STARTF(15)        POINTER TO THE FIRST VERTEX WITH THE SAME WEIGHT
C                        IN LARGE CUBE.
C      ENDF (15)          POINTER TO THE LAST VERTEX WITH THE SAME WEIGHT
C                        IN LARGE CUBE.
C *****
C      MAIN PROCEDURE
C      *
C      *
C *****
C      READ PARAMETER N, M
C
C      10 READ( 5,11,END = 130 ) N,M
C      11 FORMAT( 12,2X,I1 )
C
C      PRINT HEADING
C
C      PRINT 12
C      12 FORMAT( '1','*****' )
C      PRINT 13
C      13 FORMAT( ' ','*' )
C      PRINT 14
C      14 FORMAT( ' ','* DESIGN OF IRREDUNDANT MOS NETWORK *' )
C      PRINT 15
C      15 FORMAT( ' ','*' )
C      PRINT 16
C      16 FORMAT( ' ','*****' )
C      PRINT 17
C      17 FORMAT( '0',' X = EXTERNAL VARIABLE' )
C      PRINT 18
C      18 FORMAT( ' ',' U = OUTPUT OF MOS CELL' )
C      PRINT 19
C      19 FORMAT( '0','*****' )
C
C      CHECK PARAMETER VALUE
C
C      IF( M .EQ. 1 ) GO TO 21
C      IF( M .EQ. 2 ) GO TO 22
C      IF( M .EQ. 3 ) GO TO 23
C      IF( M .EQ. 4 ) GO TO 24
C      GO TO 25
C      21 IF( N.GE.1 .AND. N.LE.10 ) GO TO 30
C      GO TO 25
C      22 IF( N.GE.1 .AND. N.LE.9 ) GO TO 30
C      GO TO 25
C      23 IF( N.GE.1 .AND. N.LE.8 ) GO TO 30
C      GO TO 25
C      24 IF( N.GE.1 .AND. N.LE.7 ) GO TO 30
C
C      ERROR IN PARAMETER VALUE
C
C      25 PRINT 26,N,M
C      26 FORMAT( '0','INPUT ERROR IN PARAMETER CARD',3X,'N =',I3,'M =',I3 )
C      GO TO 130

```

```

C   PRINT PARAMETER VALUE
C
30 PRINT 31,N
31 FORMAT( 1-,'NUMBER OF EXTERNAL VARIABLES =',I3 )
   PRINT 32,M
32 FORMAT( 1-,'NUMBER OF OUTPUT FUNCTIONS =',I3 )
C
C   INITIALIZE LABEL( 2**N ) AND DCARE( 2**N )
C
   NVTEXL = 2**N
   DO 50 I = 1,NVTEXL
       LABEL( I ) = 0
       DCARE( I ) = 0
50 CONTINUE
   CALL INPUT( CS,&130 )
C
C   SET TIMER
C
   CALL STEPZ( TIME )
   CALL NCUBE
   II = 1
   RF = 16
   TFET = 0
   PRINT 51
51 FORMAT( 1-,'NETWORK CONFIGURATION' )
60 CALL CMNL
   IF( RF .EQ. 1 ) GO TO 90
   CALL CMXL
C
C   IF MNL( I ) = MXL( I ) FOR EVERY I IN N-CUBE, CMNL, CMXL,MPF
C   CAN BE SKIPPED.
C
   DO 70 I = 1,NVTEXL
       IF( MNL( I ) .NE. MXL( I ) ) GO TO 80
70 CONTINUE
   GO TO 120
80 CALL MPF
   CALL IMC( TFET,&10 )
   II = II + 1
   IF( CS ) GO TO 100
   IF( II .NE. RF ) GO TO 60
90 CALL ASFUN1
   CALL IMC( TFET,&10 )
   GO TO 125
100 IF( RF - II + 1 .GT. M ) GO TO 60
C
C   DECIDE THE MOS CELL CONFIGURATIONS IN COMPLETELY SPECIFIED
C   FUNCTION PART
C
110 CALL ASFUN1
   CALL IMC( TFET,&10 )
   II = II + 1
   IF( II .LE. RF ) GO TO 110
   GO TO 125
C
C   MNL( I ) = MXL( I ) OCCURED FOR EVERY I IN N-CUBE
C
120 CALL ASFUN2
   CALL IMC( TFET,&10 )
   II = II + 1
   IF( II .LE. RF ) GO TO 120

```

```

C
C PRINT STATISTICS
C
125 PRINT 126,RF
126 FORMAT( ' ', 'NUMBER OF MOS CELLS =', I3 )
127 PRINT 127,TFET
128 FORMAT( ' ', 'NUMBER OF MOS FETS =', I3 )
129 PRINT 129
129 FORMAT( ' ', '( WITHOUT FACTORING )' )
CALL STEPZ( ITIME )
UTIME = ( TIME - ITIME ) / 100.
128 PRINT 128,UTIME
128 FORMAT( '0', 'ELAPSED TIME =', F7.2, ' SEC' )
GO TO 10
130 STOP
END

C *****
C SUBROUTINE NCUBE
C *****
C SUBROUTINE NCUBE
C THIS SUBROUTINE CONSTRUCTS NCUBE ACCORDING TO THE NUMBER OF EXTERNAL
C VARIABLES. CHECK THE INPUT VECTOR IN BINARY FORM ASSIGNED TO EACH
C VERTEX ONE BY ONE AND LINKS THE VERTICES WITH THE SAME WEIGHT.
  IMPLICIT INTEGER( A-Z )
  INTEGER*2 LABEL, DCARE, MNL, MXL, CHAIN, FUNC, LINK
  1 COMMON, MINV, N, M, RF
  1 LABEL(1024), DCARE(1024), MNL(1024), MXL(1024)
  2 CHAIN(1024), STARTL(11), ENDL(11), FUNC(16384)
  3 LINK(16384), MINV(16384), STARTF(15), ENDF(15)
  MXVTXL = 2*N - 1
  L = N + 1
  DO 10 I = 1, L
  10 STARTL( I ) = 0
  I = 0
  20 X = 1
  W = 0
  30 IF( X.EQ.0 ) GO TO 50
  IF( X/2*2.EQ.X ) GO TO 40
  W = W + 1
  40 X = X/2
  GO TO 30
  50 IF( STARTL( W + 1 ).EQ.0 ) GO TO 60
  CHAIN( ENDL( W + 1 ) ) = I + 1
  ENDL( W + 1 ) = I + 1
  GO TO 70
  60 STARTL( W + 1 ) = I + 1
  ENDL( W + 1 ) = I + 1
  70 I = I + 1
  IF( I.LE.MXVTXL ) GO TO 20
  RETURN
  END

C *****
C SUBROUTINE INPUT
C *****
C SUBROUTINE INPUT( CS,* )
C THIS SUBROUTINE READS DATA INTO LABEL( 2*N ) AND DCARE( 2*N )

```

```

C   ON INPUT ERROR, THE FOLLOWING MESSAGE IS PRINTED OUT AND EXECUTION
C   IS TERMINATED.
C   I = FUNCTION NO.  J = CARD NO.  K = COLUMN NO.
C   IMPLICIT INTEGER( A-Z )
C   INTEGER*2 LABEL ,DCARE ,MNL ,MXL ,CHAIN ,FUNC ,LINK
C   1 COMMON ,MINV ,N ,M ,RF
C   1 ,LABEL(1024) ,DCARE(1024) ,MNL(1024) ,MXL(1024)
C   2 ,CHAIN(1024) ,STARTL(11) ,ENOL(11) ,FUNC(16384)
C   3 ,LINK(16384) ,MINV(16384) ,STARTF(15) ,ENDF(15)
C   LOGICAL CS
C   INTEGER*2 CHAR(80) ,ZERO/'0'/,ONE/'1'/,DCR/'*'/
C   INTEGER*2 BLANK/' '/
C   CHAR( 80 ) CHARACTER BUFFER FOR ONE CARD.
C   CS IS SET IF THE GIVEN FUNCTIONS ARE COMPLETELY
C   SPECIFIED.
C   CS = .TRUE.
C   IF( 2**N/80*80 .EQ. 2**N ) GO TO 10
C   NCAPD = 2**N/80 + 1
C   GO TO 20
10  NCAPD = 2**N/80
20  DO 140 I = 1,M
C   PRINT 30,I
30  FORMAT( '0','FUNCTION',I3 )
C   VTEX = 1
C   DO 100 J = 1,NCARD
40  READ 40,CHAR
C   FORMAT( 80A1 )
C   DO 80 K = 1,80
C   IF( CHAR( K ) .EQ. BLANK ) GO TO 110
C   IF( CHAR( K ) .EQ. ZERO ) GO TO 70
C   IF( CHAR( K ) .EQ. ONE ) GO TO 50
C   IF( CHAR( K ) .EQ. DCR ) GO TO 60
C   GO TO 150
50  LABEL( VTEX ) = LABEL( VTEX ) + 2**( M - I )
C   GO TO 70
60  DCARE( VTEX ) = DCARE( VTEX ) + 2**( M - I )
C   CS = .FALSE.
C   VTEX = VTEX + 1
80  CONTINUE
C   PRINT 90,CHAR
90  FORMAT( ' ',3X,80A1 )
100 CONTINUE
C   GO TO 120
110 IF( J .NE. NCARD ) GO TO 150
C   IF( VTEX .NE. 2**N + 1 ) GO TO 150
120 PRINT 130,CHAR
130 FORMAT( ' ',80A1 )
140 CONTINUE
C   GO TO 170
150 PRINT 160,I,J,K
160 FORMAT( '0','INPUT ERROR IN DATA CARD',3X,'I =',I3,'J =',I3,
1  'K =',I3 )
C   RETURN
170 RETURN
C   END
C *****
C SUBROUTINE CMNL
C *****

```

```

      SUBROUTINE CMNL
C     THIS SUBROUTINE IMPLEMENTS CONDITIONAL MINIMUM LABELING.
C     THIS SUBROUTINE CALLS SUBROUTINE-DSCAN AND SUBROUTINE-INCMNT.
      IMPLICIT INTEGER( A-Z )
      INTEGER*2 LABEL ,DCARE ,MNL ,MXL ,CHAIN ,FUNC ,LINK
      1 COMMON ,MINV
      1 COMMON II ,N ,M ,RF
      1 ,LABEL(1024) ,DCARE(1024) ,MNL(1024) ,MXL(1024)
      2 ,CHAIN(1024) ,STARTL(11) ,ENDL(11) ,FUNC(16384)
      3 ,LINK(16384) ,MINV(16384) ,STARTF(15) ,ENDF(15)
      INTEGER BWEIT(4)
C     PTRB POINTS TO THE VERTEX-B FOR WHICH WE ARE SEEKING FOR THE
C     MINIMUM POSSIBLE LABEL.
C     PTRB POINTS TO THE VERTEX-A WHICH HAS BIGGER WEIGHT BY ONE THAN
C     VERTEX-B AND CONNECTED TO VERTEX-B BY THE EDGE IN N-CUBE.
C     RF( MINIMUM NUMBER OF MOS CELLS ) IS OBTAINED IN THE FIRST EXECUTION
C     OF THIS SUBROUTINE.
      F = M
      MNL( STARTL( N + 1 ) ) = LABEL( STARTL( N + 1 ) )
      USPFY = RF - II + 1
      W = N
10  PTRB = STARTL( W )
20  LBX = LABEL( PTRB )
      LBY = DCARE( PTRB )
      INC = 0
      SHIFT = 1
      CALL DSCAN( LBY,USPFY,ND CARE,BWEIT )
      DO 50 I = 1,N
        PTRBX = ( PTRB - 1 ) / SHIFT
        IF( PTRBX/2*2 .NE. PTRBX ) GO TO 40
        PTRB = PTRB + SHIFT
        LAX = MNL( PTRB )
30  IF( LBX .GE. LAX ) GO TO 40
        LBX = LABEL( PTRB )
        CALL INCMNT( INC,ND CARE,LBX,F,BWEIT )
        GO TO 30
40  SHIFT = 2 * SHIFT
50  CONTINUE
      MNL( PTRB ) = LBX
      IF( PTRB .EQ. ENDL( W ) ) GO TO 60
      PTRB = CHAIN( PTRB )
      GO TO 20
60  W = W - 1
      IF( W .GE. 1 ) GO TO 10
      IF( II .NE. 1 ) GO TO 100
      RF = 0
      MNLX = MNL( 1 )
70  IF( MNLX .EQ. 0 ) GO TO 100
      MNLX = MNLX / 2
      RF = RF + 1
      GO TO 70
100 RETURN
      END
C *****
C     SUBROUTINE CMXL
C *****
C     SUBROUTINE CMXL
C     THIS SUBROUTINE IMPLEMENTS CONDITIONAL MAXIMUM LABELING.
C     THIS SUBROUTINE CALLS SUBROUTINE-DSCAN, SUBROUTINE-ASIGN1 AND

```

```

C SUBROUTINE-DCRMNT.
  IMPLICIT INTEGER( A-Z )
  INTEGER*2 LABEL ,DCARE ,MNL ,MXL ,CHAIN ,FUNC ,LINK
  1 COMMON ,MINV
  1 ,LABEL(1024) ,DCARE(1024) ,MNL(1024) ,MXL(1024)
  2 ,CHAIN(1024) ,STARTL(11) ,ENDL(11) ,FUNC(16384)
  3 ,LINK(16384) ,MINV(16384) ,STARTF(15) ,ENDF(15)
  INTEGER BWEIT(4)
C PTRB POINTS TO THE VERTEX-B FOR WHICH WE ARE SEEKING FOR THE
C MAXIMUM POSSIBLE LABEL.
C PTRB POINTS TO THE VERTEX-A WHICH HAS SMALLER WEIGHT BY ONE THAN
C VERTEX-B AND CONNECTED TO VERTEX-B BY THE EDGE IN N-CUBE.
  F = M
  LBX = LABEL( STARTL( 1 ) )
  LBY = DCARE( STARTL( 1 ) )
  USPFY = RF - 11 + 1
  CALL ASIGN1( USPFY,F,LBX,LBY )
  MXL( STARTL( 1 ) ) = LBX
  L = N + 1
  DO 60 W = 2,L
    PTRB = STARTL( W )
    20 LBY = DCARE( PTRB )
    CALL DSCAN( LBY,USPFY,NDCARE,BWEIT )
    LBX = LABEL( PTRB )
    LBY = DCARE( PTRB )
    CALL ASIGN1( USPFY,F,LBX,LBY )
    LBXX = LBX
    INC = 0
    SHIFT = 1
    DO 50 I = 1,N
      PTRBX = ( PTRB - 1 ) / SHIFT
      IF( PTRBX/2*2 .EQ. PTRBX ) GO TO 40
      PTRB = PTRB - SHIFT
      LAX = MXL( PTRB )
      30 IF( LBXX .LE. LAX ) GO TO 40
      LBXX = LBX
      CALL DCRMNT( INC,NDCARE,LBXX,F,BWEIT )
      GO TO 30
    40 SHIFT = 2 * SHIFT
    50 CONTINUE
    MXL( PTRB ) = LBXX
    IF( PTRB .EQ. ENDL( W ) ) GO TO 60
    PTRB = CHAIN( PTRB )
    GO TO 20
  60 CONTINUE
  RETURN
  END
C *****
C SUBROUTINE DSCAN
C *****
C SUBROUTINE DSCAN( LBY,USPFY,NDCARE,BWEIT )
C THIS SUBROUTINE SCAN THE DCARE WHICH IS ASSIGNED TO EACH VERTEX
C AND OBTAIN THE NUMBER OF DONTCARE BITS AND THE WEIGHT ASSIGNED
C TO EACH DONTCARE BIT.
C IMPLICIT INTEGER( A-Z )
C INTEGER BWEIT(4)
C PWFIT( 4 ) STORES THE WEIGHT ASSIGNED TO EACH DONTCARE BIT.
C NDCARE STORES THE NUMBER OF DONTCARE BITS IN ONE VERTEX.

```

```

NDCARE = 0
I = 1
SHIFT = 1
DO 20 I = 1,USPFY
  IF( LBY.EQ. 0 ) GO TO 30
  IF( LBY/2*2.EQ. LBY ) GO TO 10
  NDCARE = NDCARE + 1
  BWEIT( NDCARE ) = SHIFT
10  SHIFT = 2 * SHIFT
  LBY = LBY / 2
20 CONTINUE
30 RETURN
END

C *****
C SUBROUTINE INCMNT *
C *****
C SUBROUTINE INCMNT( INC,NDCARE,LBX,M,BWEIT )
C THIS SUBROUTINE INCREMENTS THE LABEL ASSIGNED TO THE VERTEX POINTED
C BY POINTER-PTRB.
C IMPLICIT INTEGER( A-Z )
C INTEGER BWEIT( 4 )
C INC IS THE COUNTER WHICH HAS SPECIFIED WEIGHT FOR EACH BIT.
  INC = INC + 1
  INCX = INC
  J = 1
  SHIFT = 2 ** ( M - NDCARE )
10 IF( INCX.EQ. 0 ) GO TO 40
  IF( INCX/2*2.EQ. INCX ) GO TO 30
  IF( J.LE. NDCARE ) GO TO 20
  LBX = LBX + SHIFT
  GO TO 30
20 LBX = LBX + BWEIT( J )
30 J = J + 1
  SHIFT = 2 * SHIFT
  INCX = INCX / 2
  GO TO 10
40 RETURN
END

C *****
C SUBROUTINE ASIGN1 *
C *****
C SUBROUTINE ASIGN1( USPFY,M,LBX,LBY )
C THIS SUBROUTINE ASSIGNS ONE TO THE EVERY UNSPECIFIED BIT OF LABELS
C IN N-CUBE.
C IMPLICIT INTEGER( A-Z )
  SHIFT = 1
  IF( USPFY.GT.M ) GO TO 30
  DO 20 I = 1,USPFY
    IF( LBY/2*2.EQ. LBY ) GO TO 10
    LBX = LBX + SHIFT
10  SHIFT = 2 * SHIFT
    LBY = LBY / 2
20 CONTINUE
  GO TO 70
30 DO 50 I = 1,M
  IF( LBY/2*2.EQ. LBY ) GO TO 40
  LBX = LBX + SHIFT

```

```

40     SHIFT = 2 * SHIFT
      LBY = LBY / 2
50     CONTINUE
      L = M + 1
      DO 60 I = L,USPFY
          LBX = LBX + SHIFT
          SHIFT = 2 * SHIFT
60     CONTINUE
70     RETURN
      END
C *****
C SUBROUTINE DCRMNT
C *****
C SUBROUTINE DCRMNT( INC,ND CARE,LBXX,M,BWEIT )
C THIS SUBROUTINE DECREMENT THE LABEL ASSIGNED TO THE VERTEX POINTED
C BY POINTER-PTRB
      IMPLICIT INTEGER( A-Z )
      INTEGER BWEIT( 4 )
      INC = INC + 1
      INCX = INC
      J = 1
      SHIFT = 2 ** ( M - ND CARE )
10     IF( INCX.EQ.0 ) GO TO 40
      IF( INCX/2*2 .EQ. INCX ) GO TO 30
      IF( J.LE.ND CARE ) GO TO 20
      LBXX = LBXX - SHIFT
      GO TO 30
20     LBXX = LBXX - BWEIT( J )
30     J = J + 1
      SHIFT = 2 * SHIFT
      INCX = INCX / 2
      GO TO 10
40     RETURN
      END
C *****
C SUBROUTINE MPF
C *****
C SUBROUTINE MPF
C THIS SUBROUTINE OBTAINS MAXIMUM PERMISSIBLE FUNCTION FROM MINIMUM
C LABEL AND MAXIMUM LABEL AND STORES THE RESULT TO FUNC( NVTEFX ).
      IMPLICIT INTEGER( A-Z )
      INTEGER*2 LABEL ,DCARE ,MNL ,MXL ,CHAIN ,FUNC ,LINK
1     COMMON ,II ,N ,M ,RF
1     ,LABEL(1024) ,DCARE(1024) ,MNL(1024) ,MXL(1024)
2     ,CHAIN(1024) ,STARTL(11) ,ENOL(11) ,FUNC(16384)
3     ,LINK(16384) ,MINV(16384) ,STARTF(15) ,ENDF(15)
      MXVTXL = 2**N - 1
      NVTEFX = 2** ( N + II - 1 )
      RSHIFT = 2** ( RF - II )
      LSHIFT = 2** ( II - 1 )
      DO 10 I = 1,NVTEFX
10     FUNC( I ) = 0
      I = 0
20     MNLX = MNL( I + 1 ) / RSHIFT
      MXLX = MXL( I + 1 ) / RSHIFT
      J = I*LSHIFT + MNLX/2

```

```

      IF( MNLX/2*2 .EQ. MNLX ) GO TO 30
      IF( MXLX/2*2 .EQ. MXLX ) GO TO 40
      FUNC( J + 1 ) = 2
      GO TO 40
30  IF( MXLX/2*2 .NE. MXLX ) GO TO 40
      FUNC( J + 1 ) = 1
40  I = I + 1
      IF( I.LE.MXVTXL ) GO TO 20
      RETURN
      END
C *****
C      SUBROUTINE IMC      *
C      *                  *
C *****
C      SUBROUTINE IMC( TFET,* )
C      THIS SUBROUTINE OBTAINS IRREDUNDANT MOS CELL CONFIGURATION FROM
C      THE MAXIMUM PERMISSIBLE FUNCTION.
C      THIS SUBROUTINE IS MADE OF THE FOLLOWING SIX STEPS.
C      STEP-1  CONSTRUCTS LARGE-CUBE FUNC( 2**(N + II - 1) ).
C      STEP-2  ASSIGNS "0" OR "1" TO THE VERTEX-X WITH DONT CARE SUCH
C               THAT THERE EXISTS NO VERTEX-Y SATISFYING Y > X AND
C               F( Y ) = "1".
C      STEP-3  OBTAINS THE SET OF MINIMUM VECTORS.
C      STEP-4  OBTAINS THE SUBSET OF THE SET OF MINIMUM VECTORS WHICH
C               COVERS EVERY VERTEX WITH ORIGINAL "0".
C      STEP-5  OBTAINS AN IRREDUNDANT SUBSET FROM THE SUBSET OBTAINED
C               IN STEP-4.
C      STEP-6  STORES THE RESULT TO THE VERTEX IN N-CUBE ( LABEL ).
C               THIS STEP CONTAINS ERROR CHECKING ROUTINE.
C               IF FUNCTION VALUES FROM MOS CELL CONFIGURATION OBTAINED
C               IN PREVIOUS STEPS ARE DIFFERENT FROM THE FUNCTION VALUES
C               ALREADY STORED IN ARRAY-LABEL, ERROR MESSAGE AND
C               THE CONTENTS OF ARRAY-LABEL IS PRINTED.
C      IMPLICIT INTEGER( A-Z )
C      INTEGER*2 LABEL ,DCARE ,MNL ,MXL ,CHAIN ,FUNC ,LINK
C      1 COMMON ,MINV
C      1 ,LABEL(1024) ,N ,M ,RF
C      2 ,CHAIN(1024) ,DCARE(1024) ,MNL(1024) ,MXL(1024)
C      3 ,LINK(16384) ,STARTL(11) ,ENDL(11) ,FUNC(16384)
C      ,MINV(16384) ,STARTF(15) ,ENDF(15)
C      INTEGER X( 10 ) / :--X1', :--X2', :--X3', :--X4', :--X5',
C      1 :--X6', :--X7', :--X8', :--X9', :--X10', / ,
C      2 U( 10 ) / :--U1', :--U2', :--U3', :--U4', :--U5', / ,
C      3 :--U6', :--U7', :--U8', :--U9', :--U10', /
C      INTEGER BLANK/ ' / , FORM1/ ' / , FORM2/ '0--1' / ,
C      1 FORM3/ :--1' / , FORM4/ '1' / , FORM5/ :--1' / ,
C      2 FORM6/ '0' / , FORM7/ :--1' / , FORM8/ '1' / ,
C      3 FORM9/ '0--1' /
C      INTEGER PRTRY( 20 )
C      LOGICAL ERROR
C      LOGICAL REPEAT
C      PRTRY( 20 ) PRINT BUFFER FOR PRINTING OBTAINED MOS GATE
C      CONFIGURATION.
C      ERROR IS SET IF ERROR OCCURS IN CONSTRUCTING NETWORK.
C      INPD = N + II - 1
C      NVTEXL = 2 ** N
C      MXVTXL = NVTEXL - 1
C      NVTEXF = 2 ** INPD
C      MXVTXF = NVTEXF - 1
C      LSHIFT = 2 ** ( II - 1 )

```

```

      RSHIFT = 2 ** ( RF - II )
      ERROR = .FALSE.
C
C   STEP-1
      L = INPD + 1
      DO 10 I = 1, L
10    STARTF( I ) = 0
      I = 0
20    Y = I
      W = 0
30    IF( Y .EQ. 0 ) GO TO 50
      IF( Y/2*2 .EQ. Y ) GO TO 40
      W = W + 1
40    Y = Y / 2
      GO TO 30
50    IF( STARTF( W + 1 ) .EQ. 0 ) GO TO 60
      LINK( FENDF( W + 1 ) ) = I + 1
      ENDF( W + 1 ) = I + 1
      GO TO 70
60    STARTF( W + 1 ) = I + 1
      ENDF( W + 1 ) = I + 1
70    I = I + 1
      IF( I .LE. MXVTXF ) GO TO 20
C
C   STEP-2
      IF( FUNC( STARTF( INPD + 1 ) ) .NE. 0 ) GO TO 110
      FUNC( STARTF( INPD + 1 ) ) = 3
110   W = INPD
120   PTRB = STARTF( W )
130   IF( FUNC( PTRB ) .NE. 0 ) GO TO 160
      SHIFT = 1
      DO 150 I = 1, INPD
          PTRBX = ( PTRB - 1 ) / SHIFT
          IF( PTRBX/2*2 .NE. PTRBX ) GO TO 140
          PTRB = PTRB + SHIFT
          IF( FUNC( PTRB ) .NE. 2 ) GO TO 140
          FUNC( PTRB ) = 2
          GO TO 160
140   SHIFT = 2 * SHIFT
150   CONTINUE
      FUNC( PTRB ) = 3
160   IF( PTRB .EQ. ENDF( W ) ) GO TO 170
      PTRB = LINK( PTRB )
      GO TO 130
170   W = W - 1
      IF( W .GE. 1 ) GO TO 120
C
C   STEP-3
      K = 1
      NMINV = 0
      W = INPD + 1
210   PTRB = STARTF( W )
220   IF( FUNC( PTRB ) .EQ. 2 ) GO TO 250
      SHIFT = 1
      DO 240 I = 1, INPD
          PTRBX = ( PTRB - 1 ) / SHIFT
          IF( PTRBX/2*2 .EQ. PTRBX ) GO TO 230
          PTRB = PTRB - SHIFT

```

```

      IF( FUNC( PTRB ).NE.2 ) GO TO 250
230  SHIFT = 2 * SHIFT
240  CONTINUE
      MINV( K ) = PTRB - 1
      NMINV = NMINV + 1
      K = K + 1
250  IF( PTRB.EQ.ENDF( W ) ) GO TO 260
      PTRB = LINK( PTRB )
      GO TO 220
260  W = W - 1
      IF( W.GE.1 ) GO TO 210

C
C  STEP-4
C
      DO 310 I = 1,NVTEF
310  LINK( I ) = 0
      I = 1
320  REPEAT = .FALSE.
      DO 340 J = 1,NVTEF
          IF( FUNC( J ).NE.1 ) GO TO 340
          JX = J - 1
          MINX = MINV( I )
          CALL COMPR( JX,MINX,&330 )
          LINK( J ) = LINK( J ) + 1
          GO TO 340
330  IF( LINK( J ).NE.0 ) GO TO 340
          REPEAT = .TRUE.
340  CONTINUE
      IF( REPEAT ) GO TO 350
      NSIRR = I
      GO TO 360
350  I = I + 1
      GO TO 320
360  CONTINUE

C
C  STEP-5
C
      L = NSIRR - 1
      DO 450 I = 1,L
          DO 410 J = 1,NVTEF
              IF( FUNC( J ).NE.1 ) GO TO 410
              JX = J - 1
              MINX = MINV( I )
              CALL COMPR( JX,MINX,&410 )
              LINK( J ) = LINK( J ) - 1
              IF( LINK( J ).EQ.0 ) GO TO 420
410  CONTINUE
          GO TO 440
420  IF( FUNC( J ).NE.1 ) GO TO 430
          JX = J - 1
          MINX = MINV( I )
          CALL COMPR( JX,MINX,&430 )
          LINK( J ) = LINK( J ) + 1
430  J = J - 1
          IF( J.GE.1 ) GO TO 420
          GO TO 450
440  MINV( I ) = -1
450  CONTINUE

C
C  PRINT MOS GATE CONFIGURATION
C

```

```

NIRR = 0
MXFET = 0
DO 490 I = 1, NSIRR
  IF( MINV( I ) .LT. 0 ) GO TO 490
  NIRR = NIRR + 1
  MINX = MINV( I )
  NFET = 0
450 IF( MINX .EQ. 0 ) GO TO 480
  IF( MINX/2*2 .EQ. MINX ) GO TO 470
  NFET = NFET + 1
  MINX = MINX / 2
  GO TO 460
480 TFET = TFET + NFET
  IF( NFET .LE. MXFET ) GO TO 490
  MXFET = NFET
490 CONTINUE
  PRINT 4100
+100 FORMAT( '0' )
  I = 0
  L = 2 * NIRR - 1
  DO 4290 LINE = 1, L
    IF( LINE/2*2 .EQ. LINE ) GO TO 4240
C
C   PRINT ODD LINE
C
4101 I = I + 1
    IF( MINV( I ) .LT. 0 ) GO TO 4101
    IF( LINE .EQ. NIRR ) GO TO 4110
    PRTRY( 1 ) = FORM1
    GO TO 4120
4110 IF( NIRR .EQ. 1 ) GO TO 4111
    PRTRY( 1 ) = FORM2
    GO TO 4120
4111 PRTRY( 1 ) = FORM9
4120 K = 0
    SHIFT = 2 ** ( N + II - 2 )
    DO 4140 J = 1, N
      MINX = MINV( I ) / SHIFT
      IF( MINX/2*2 .EQ. MINX ) GO TO 4130
      K = K + 1
      PRTRY( K + 1 ) = X( J )
      SHIFT = SHIFT / 2
4130 CONTINUE
4140 IF( II .EQ. 1 ) GO TO 4170
      N1 = N + 1
      N2 = N + II - 1
      DO 4160 J = N1, N2
        MINX = MINV( I ) / SHIFT
        IF( MINX/2*2 .EQ. MINX ) GO TO 4150
        K = K + 1
        PRTRY( K + 1 ) = U( J - N )
        SHIFT = SHIFT / 2
4150 CONTINUE
4160 K = K + 1
4170 IF( K .GT. MXFET ) GO TO 4180
      PRTRY( K + 1 ) = FORM7
      GO TO 4170
4180 IF( LINE .EQ. NIRR ) GO TO 4210
      PRTRY( MXFET + 2 ) = FORM3
      PRTRY( MXFET + 3 ) = BLANK
4190 MX3 = MXFET + 3

```

```

      PRINT 4200,( PRTARY( JJ ),JJ = 1,MX3 )
4200  FORMAT( ' ',12X,20A4 )
      GO TO 4290
4210  IF( NIRR .EQ. 1 ) GO TO 4211
      PRTARY( MXFET + 2 ) = FORM5
      PRTARY( MXFET + 3 ) = FORM6
      GO TO 4220
4211  PRTARY( MXFET + 2 ) = FORM7
      PRTARY( MXFET + 3 ) = FORM6
4220  MX3 = MXFET + 3
      PRINT 4230,II,( PRTARY( JJ ),JJ = 1,MX3 )
4230  FORMAT( ' ',MOS CELL',12,2X,20A4 )
      GO TO 4290

```

C
C PRINT EVEN LINE
C

```

4240  IF( LINE .EQ. NIRR ) GO TO 4250
      PRTARY( 1 ) = FORM1
      GO TO 4260
4250  PRTARY( 1 ) = FORM2
4260  DO 4270 J = 1,MXFET
4270  PRTARY( J + 1 ) = BLANK
      IF( LINE .EQ. NIRR ) GO TO 4280
      PRTARY( MXFET + 2 ) = FORM4
      PRTARY( MXFET + 3 ) = BLANK
      GO TO 4190
4230  PRTARY( MXFET + 2 ) = FORM8
      PRTARY( MXFET + 3 ) = FORM6
      GO TO 4220
4290  CONTINUE
      IF( NIRR .EQ. NMINV ) GO TO 610

```

C
C STEP-6
C

THE SIZE OF IRREDUNDANT SUBSET IS NOT EQUAL TO THE SIZE OF
MINIMUM VECTOR SET.

C
C
C
C
C
C

```

      I = 0
510  J = I*LSHIFT + LABEL( I + 1 )/( 2 * RSHIFT )
      DO 520 K = 1,NSIRR
      IF( MINV( K ) .LT. 0 ) GO TO 520
      JX = J
      MINX = MINV( K )
      CALL COMPR( JX,MINX,&520 )
      GO TO 540
520  CONTINUE
      IF( RF - II + 1 .GT. M ) GO TO 530
      DCARX = DCARE( I + 1 ) / RSHIFT
      IF( DCARX/2*2 .NE. DCARX ) GO TO 530
      LABELX = LABEL( I + 1 ) / RSHIFT
      IF( LABELX/2*2 .NE. LABELX ) GO TO 550
      ERROR = .TRUE.
      GO TO 550
530  LABEL( I + 1 ) = LABEL( I + 1 ) + RSHIFT
      GO TO 550
540  IF( RF - II + 1 .GT. M ) GO TO 550
      DCARX = DCARE( I + 1 ) / RSHIFT
      IF( DCARX/2*2 .NE. DCARX ) GO TO 550
      LABELX = LABEL( I + 1 ) / RSHIFT
      IF( LABELX/2*2 .EQ. LABELX ) GO TO 550
      ERROR = .TRUE.

```

```

550 I = I + 1
    IF( I .LE. MXVTXL ) GO TO 510
    GO TO 660
C
C   THE SIZE OF IRREDUNDANT SUBSET IS EQUAL TO THE SIZE OF
C   MINIMUM VECTOR SET.
C
610 I = 0
620 J = I*LSHIFT + LABEL( I + 1 )/( 2 * RSHIFT )
    IF( FUNC( J + 1 ) .EQ. 2 ) GO TO 630
    IF( RF - II + 1 .GT. M ) GO TO 650
    DCARX = DCARE( I + 1 ) / RSHIFT
    IF( DCARX/2*2 .NE. DCARX ) GO TO 650
    LABELX = LABEL( I + 1 ) / RSHIFT
    IF( LABELX/2*2 .EQ. LABELX ) GO TO 650
    ERROR = .TRUE.
    GO TO 650
630 IF( RF - II + 1 .GT. M ) GO TO 640
    DCARX = DCARE( I + 1 ) / RSHIFT
    IF( DCARX/2*2 .NE. DCARX ) GO TO 640
    LABELX = LABEL( I + 1 ) / RSHIFT
    IF( LABELX/2*2 .NE. LABELX ) GO TO 650
    ERROR = .TRUE.
    GO TO 650
640 LABEL( I + 1 ) = LABEL( I + 1 ) + RSHIFT
650 I = I + 1
    IF( I .LE. MXVTXL ) GO TO 620
660 IF( ERROR ) GO TO 670
    GO TO 690
670 PRINT 680,II
680 FORMAT( '0','ERROR IN CONSTRUCTING MOS GATE',I3 )
    CALL PRINT( LABEL,NVTEXTL )
    RETURN
690 RETURN
    END
C *****
C
C   SUBROUTINE COMPR
C
C *****
C   SUBROUTINE COMPR( JX,MINX,* )
C   THIS SUBROUTINE COMPARES THE SIZE OF TWO VECTORS JX AND MINX
10 IF( MINX/2*2 .EQ. MINX ) GO TO 20
    IF( JX/2*2 .EQ. JX ) RETURN
20 JX = JX / 2
    MINX = MINX / 2
    IF( JX.NE.0 .OR. MINX.NE.0 ) GO TO 10
    RETURN
    END
C *****
C
C   SUBROUTINE ASFUN1
C
C *****
C   SUBROUTINE ASFUN1
C   THIS SUBROUTINE ASSIGNS THE LABEL OR FUNCTION VALUE FROM N-CUBE
C   TO LAPGE-CUBE( FUNC ).
    IMPLICIT INTEGER( A-Z )
    INTEGER*2 LABEL ,DCARE ,MNL ,MXL ,CHAIN ,FUNC ,LINK
1    ,MINV
    COMMON II ,N ,M ,RF

```

```

1  ,LABEL(1024)      ,DCARE(1024)      ,MNL(1024)      ,MXL(1024)
2  ,CHAIN(1024)      ,STARTL(11)      ,ENDL(11)      ,FUNC(16384)
3  ,LINK(16384)      ,MINV(16384)      ,STARTF(15)      ,ENDF(15)
  MXVTXL = 2**N-1
  NVTEXF = 2** ( N + II - 1 )
  RSHIFT = 2** ( RF - II )
  LSHIFT = 2** ( II - 1 )
  DO 10 I = 1,NVTEXF
10 FUNC( I ) = 0
  I = 0
20 LABELX = LABEL( I + 1 ) / RSHIFT
  DCARX = DCARE( I + 1 ) / RSHIFT
  J = I*LSHIFT + LABELX/2
  IF( DCARX/2*2 .NE. DCARX ) GO TO 40
  IF( LABELX/2*2 .EQ. LABELX ) GO TO 30
  FUNC( J + 1 ) = 2
  GO TO 40
30 FUNC( J + 1 ) = 1
40 I = I + 1
  IF( I.LE.MXVTXL ) GO TO 20
  RETURN
END
C *****
C SUBROUTINE ASFUN2
C *****
C SUBROUTINE ASFUN2
C THIS SUBROUTINE ASSIGNS THE LABEL OR FUNCTION VALUE FROM
C MNL( N-CUBE ) TO LARGE-CUBE( FUNC ).
  IMPLICIT INTEGER( A-Z )
  INTEGER*2 LABEL ,DCARE ,MNL ,MXL ,CHAIN ,FUNC ,LINK
1  COMMON II ,N ,M ,RF
1  ,LABEL(1024) ,DCARE(1024) ,MNL(1024) ,MXL(1024)
2  ,CHAIN(1024) ,STARTL(11) ,ENDL(11) ,FUNC(16384)
3  ,LINK(16384) ,MINV(16384) ,STARTF(15) ,ENDF(15)
  MXVTXL = 2**N - 1
  NVTEXF = 2** ( N + II - 1 )
  RSHIFT = 2** ( RF - II )
  LSHIFT = 2** ( II - 1 )
  DO 10 I = 1,NVTEXF
10 FUNC( I ) = 0
  I = 0
20 MNLX = MNL( I + 1 ) / RSHIFT
  J = I*LSHIFT + MNLX/2
  IF( MNLX/2*2 .EQ. MNLX ) GO TO 30
  FUNC( J + 1 ) = 2
  GO TO 40
30 FUNC( J + 1 ) = 1
40 I = I + 1
  IF( I.LE.MXVTXL ) GO TO 20
  RETURN
END
C *****
C SUBROUTINE PRINT
C *****
C THIS SUBROUTINE PRINTS THE CONTENTS OF AN ARRAY( LABEL,DCARE,ETC. ).
C ARAY IS THE ARRAY TO BE PRINTED.

```

```

C   SIZE INDICATES THE NUMBER OF THE ELEMENTS IN THE ARRAY TO BE
C   PRINTED.
      SUBROUTINE PRINT( ARRAY, SIZE )
      IMPLICIT INTEGER( A-Z )
      INTEGER*2 ARRAY(1024), PBUF(16)
      PRINT 10
10  FORMAT( '0', '*****DUMP*****' )
      DO 60 I = 1, SIZE
      ARRAYX = ARRAY( I )
      J = 16
20  IF( ARRAYX/2*2 .EQ. ARRAYX ) GO TO 30
      PBUF( J ) = 1
      GO TO 40
30  PBUF( J ) = 0
40  ARRAYX = ARRAYX / 2
      J = J - 1
      IF( J.GE.1 ) GO TO 20
      PRINT 50, ( PBUF( JJ ), JJ = 1, 16 )
50  FORMAT( ',16I1' )
60  CONTINUE
      RETURN
      END

```

1. Report No. UIUCDCS-R-76-784	2.	3. Recipient's Accession No.
4. Title and Subtitle DESIGN OF IRREDUNDANT MOS NETWORKS: A PROGRAM MANUAL FOR THE DESIGN ALGORITHM DIMN		5. Report Date February 1976
6.		7.
8. Author(s) Yazuhiko Yamamoto		9. Performing Organization Rept. No. UIUCDCS-R-76-784
10. Performing Organization Name and Address University of Illinois at Urbana-Champaign Department of Computer Science Urbana, Illinois 61801		11. Project/Task/Work Unit No.
12. Sponsoring Organization Name and Address National Science Foundation Washington, D.C.		13. Contract/Grant No. NSF Grant No. GJ-40221
14. Type of Report & Period Covered Technical		15.

16. Supplementary Notes

17. Abstracts

This paper describes a program package for the design of irredundant single- and multiple-output MOS networks. The program is an implementation of algorithms developed by H.C. Lai for the synthesis of irredundant MOS networks with a minimum number of MOS cells for a given set of completely or incompletely specified functions (synthesized networks having multiple outputs are guaranteed to have a minimum number of MOS cells only under certain assumptions). The output of the program is a graphic representation of the interconnections among FET's for each MOS cell in the synthesized irredundant network. A listing of the FORTRAN program is included in the paper.

18. Key Words and Document Analysis. 17a. Descriptors

Logic design, logic circuits, logical elements, programs (computers).

19. Identifiers/Open-Ended Terms

computer-aided-design, irredundant networks, MOS networks, optimal networks, MOS cells, MOS, FET.

20. COSATI Field/Group

21. Availability Statement RELEASE UNLIMITED	22. Security Class (This Report) UNCLASSIFIED	23. No. of Pages 125
	24. Security Class (This Page) UNCLASSIFIED	25. Price



127 6: 127

OCT 17 REC'D



UNIVERSITY OF ILLINOIS-URBANA

510.64 IL6R no. C002 no.782-787(1976

Design of Irredundant MDS Networks : s p



3 0112 088402588